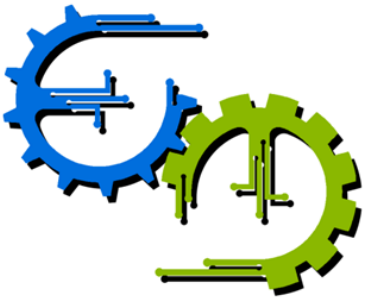
TRABALHO DE GRADUAÇÃO

LOCALIZAÇÃO DE ROBÔ HUMANOIDE APLICADO A FUTEBOL DE ROBÔS

Raphael Arthur Barbosa Resende



ENGENHARIA MECATRÔNICA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

LOCALIZAÇÃO DE ROBÔ HUMANOIDE APLICADO A FUTEBOL DE ROBÔS

Raphael Arthur Barbosa Resende

Trabalho de Graduação submetido como requisito parcial de obtenção do grau de Engenheiro de Controle e Automação.

Banca Examinadora

Prof. Antonio Padilha Lanari Bó, ENE/UnB _____
Orientador

Profª. Mariana Costa Bernardes Matias, FGA/UnB _____
Co-Orientadora

Dr. Luís Felipe da Cruz Figueredo, _____
Examinador Externo

Prof. Geovany Araújo Borges, ENE/UnB _____
Examinador Interno

Brasília, 07 de dezembro de 2016

FICHA CATALOGRÁFICA

RESENDE, RAPHAEL ARTHUR BARBOSA
Localização de Robô HumanoideAplicado a Futebol de Robôs [Distrito Federal] 2016.
xi, 66p., 210 x 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2016).
Trabalho de Graduação – Universidade de Brasília, Faculdade de Tecnologia.

1. Localização
2. RoboCup
3. Filtro de Kalman Estendido
4. Robô Humanoide NAO
I. Mecatrônica/FT/UnB
II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

RESENDE, R.A.B. (2016). Localização de Robô HumanoideAplicado a Futebol de Robôs, Trabalho de Graduação em Engenharia de Controle e Automação, Publicação TG-022/2016, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 66p.

CESSÃO DE DIREITOS

AUTOR: Raphael Arthur Barbosa Resende
TÍTULO: Localização de Robô HumanoideAplicado a Futebol de Robôs.
GRAU: Engenheiro ANO: 2016

_____

Raphael Arthur Barbosa Resende

Faculdade de Tecnologia - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Dedico este trabalho a minha mãe que sempre me apoiou.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por ter me guiado ao longo dos anos. Agradeço a minha mãe, meus avós e minhas tias por todo o apoio e as orações. Agradeço meus grandes amigos Yuri e Henrique por toda a ajuda que vocês me deram ao longo do caminho e em especial ao Jhonantans, por toda a paciência que teve comigo, além de sempre estar presente nos momentos difíceis, onde posso afirmar que sem esses três tudo que eu conquistei até agora e que ainda conquistarei não seria possível. Agradeço a todos que passaram pela UnBeatables: Cris, De Hong, Felipe, Eric, Bruno, Gabriel, Ana e Amanda, nossas viagens, zueiras, noites viradas, frustrações e vitórias vou levar por toda a minha vida. Agradeço a melhor professora de todas: Mariana Bernardes, que nunca mede seus esforços para ajudar seus alunos. Ao professor Antônio Padilha e Luís Figueredo por toda a orientação e suporte ao longo dos trabalhos desenvolvidos tanto na UnBeatables quanto neste trabalho de graduação. Também é necessário falar de meus amigos de Unaí: Bruno, Caio, Deivid e Vinícius que me acompanham desde o ensino médio e me viram passar por tudo isso e sou muito grato por tudo. E é claro que não poderia deixar de lembrar de um certo grupo de quatro aventureiros...

RESUMO

Na robótica atual, resolver o problema da localização é fundamental para o desenvolvimento de um robô autônomo, pois somente após conhecida sua posição no mundo, ele poderá realizar determinadas tarefas. No contexto do futebol de robôs humanoides é importante se posicionar de maneira adequada rumo ao gol do adversário, para que assim tenha uma real chance de fazer um gol. Neste trabalho será proposto um algoritmo baseado em filtro de Kalman estendido (FKE) para localização de um robô humanoide dentro de um campo de futebol. Serão usados dados dos sensores inerciais (girômetro e acelerômetro), além da identificação de pontos de referência a partir de sua câmera (features) para estimar a distância do robô em relação as estruturas do campo. Algumas dificuldades do ambiente de trabalho é que ele é simétrico, ou seja, as features identificáveis não são únicas, o que gera ambiguidade nos resultados e aumenta as incertezas associadas ao problema.

ABSTRACT

In robotics, solving the localization problem is critical to the development of autonomous robot, only when his position in the world is know, he can perform certain tasks. In the context of humanoid robot soccer, it is important to position yourself near to the goal of the opponent, so that you have a real chance to make a goal. In this work, an algorithm based on extended Kalman filter (EKF) will be proposed to locate a humanoid robot inside a soccer field. We will use data from the inertial sensors (gyroscope and accelerometer), in addition to of reference features observed by the camera to estimate a robot distance in relation as field structures. Some difficulties in the work environment it is that it is symmetrical, that is, the identifiable characteristics are not unique, which creates ambiguity in the results and increases the uncertainties associated with the problem.

Símbolos Latinos

$\mathbf{R}$	Matriz de rotação
p	Ponto no espaço

Símbolos Gregos

θ	Ângulo
α	Ângulo da *feature* em relação ao campo
β	Ângulo da *feature* em relação a bissetriz da câmera

Neste trabalho vetores e escalares são representados por letras minúsculas. Matrizes são representadas por letras maiúsculas.

1 INTRODUÇÃO

Na robótica, a capacidade de reconhecer sua posição no mundo é fundamental para que se possa atuar no mesmo com eficiência e pode ser estendido a diversas aplicações. A partir do instante em que é necessário andar até um local específico ou atuar em um determinado ponto, saber sua posição é fundamental.

A resolução deste problema varia conforme, não só a capacidade do robô de perceber o ambiente ao seu redor, ou seja, dos sensores que a plataforma dispõe, quanto do próprio ambiente em que ela se encontra. Por exemplo: quais cores esse ambiente apresenta, relevos, marcações, obstáculos ou qualquer outra característica que possa ser transformada em uma informação relevante para o problema.

O processo de se localizar é um assunto bem consolidado da literatura. Nele, é fundamental que se tenha um mapa bem definido, pois ele é adequado para a comunicação entre robôs e humanos, tornando disponível a informação a respeito da crença de localização, em contrapartida, caso exista alguma divergência entre o modelo e o mundo real, como erros no mapa, o robô pode apresentar comportamentos inadequados.

Outro problema associado é em relação aos sensores do robô, pois eles efetuam medidas que possuem grande dose de incerteza: as medidas são imprecisas e há ruídos. Da mesma forma que os atuadores são limitados e apresentam leituras com incertezas associadas. Estas incertezas associadas as leituras são justamente o que fazem com que a posição obtida pela navegação inercial do robô seja deslocado em relação a sua posição real no mundo. Para a resolução destes problemas é a amplamente estudada estimação bayesiana, como por exemplo o filtro de Kalman e seus variantes, bem como o filtro de partículas baseado no método de Monte Carlo.

No contexto da partida de futebol, conhecer sua posição no campo de futebol é um importante passo para que se possa planejar ações, como por exemplo, onde ir, para onde chutar, reconhecer qual é seu lado do campo e consequentemente qual é o seu gol. Sem essas informações, a atuação do robô pode ficar limitada e no pior dos casos pode levar a diversos erros, desde sair do campo, a até mesmo fazer gol contra. Assim, será abordado o desenvolvimento de um algoritmo de localização utilizando o filtro de Kalman estendido (FKE) em uma partida de futebol de robôs humanoides na competição chamada *RoboCup*, mais precisamente na categoria SPL (*Standard Plataform League*), onde também apresentaremos a implementação de um simulador 2D dotado de funcionalidades para representar esse ambiente, bem como o robô que atua nele. Onde incialmente será proposta uma estratégia de localização que seja eficaz em meio campo, uma vez que esta redução na área de trabalho simplifica o problema facilitando assim uma primeira abordagem, onde ao atingir um desempenho satisfatório iremos validar o método no campo completo e comparar os resultados obtidos.

1.1 A *ROBOCUP*

A *RoboCup* foi fundada com o intuito de incentivar pesquisas e atrair a atenção do público geral para a robótica e inteligência artificial. Estabelecendo problemas a serem resolvidos em competições nas as quais as equipes puderiam não só competir, mas também trocar informações, publicar seus trabalhos, enfim, criar uma rede de colaboradores na comunidade da robótica. Estes problemas em geral são complexos o suficientes para que se possa render frutos em pesquisas acadêmicas bem como atrair o público para estes eventos.

A meta principal da *RoboCup* é que em 2050 exista um time de robôs humanoides completamente autônomos que seja capaz de vencer uma partida de futebol sob as regras da FIFA contra o atual campeão da copa do mundo [1].

A *RoboCup* é composta de várias competições, onde citaremos brevemente as principais e entraremos em detalhes somente na *Standard Plataform League*, que é a modalidade de interesse deste trabalho. São divididas em ligas, sendo elas: *RoboCupSoccer*, *RoboCupRescue*, *RoboCup@Home*, *RoboCupIndustrial* e *RoboCupJunior*.

- *RoboCupSoccer* é umas das principais ligas do evento, tendo como foco a competição de futebol. São desenvolvidos conceitos de cooperação entre robôs e sistemas multiagentes em sistemas dinâmicos com situações adversas, onde todos os robôs nessa liga são autônomos. Dentro da liga, existem várias categorias: *Humanoide*, *Small Size*, *Middle Size*, *Simulation* e *Standard Plataform*. Uma explicação simples para a divisão de categorias, é que elas são divididas por tamanho e tipo de robôs, podendo ser humanoides ou com rodas. Nas categorias também podem ter robôs construídos pelos próprios participantes ou distribuídos como uma plataforma padrão, que é justamente o caso da *Standard Platarform*, onde é utilizado o robô NAO, da empresa *Aldebaran Robotics*, modalidade essa que inspirou este trabalho.

- A *RoboCupRescue* tem como intuito promover a pesquisa e o desenvolvimento de robôs que possam atuar em vários níveis de coordenação e trabalho em equipe para busca e resgate em desastres. Estes são sistemas muito complexos no geral, pois em algumas situações de risco pode ser necessário uma avaliação de infraestrutura e elaborar estratégias. Sendo assim, um trabalho muito delicado que exige uma plataforma robusta com estratégias flexíveis o suficiente para atuar em adversidades.

- A liga *RoboCup@Home* tem como propósito desenvolver robôs que possam atuar nas áreas de serviço e assistência a humanos, como por exemplo na atuação doméstica. Na competição é feita uma série de testes que avaliam o desempenho do robô em um ambiente real, sendo eles: Interação e cooperação humano-robô, navegação e mapeamento em ambientes dinâmicos, reconhecimento de objetos em condições naturais de iluminação, manipulação de objetos, comportamentos adaptativos.

- A *RoboCupIndustrial leagu*e é dividida em duas categorias: *@Work League* e *Logistics League*. Na primeira, a ideia é desenvolver novos robôs móveis equipados com manipuladores para atuarem em aplicações que existem e que estão por vir na indústria, onde os robôs devem cooperar com trabalhadores humanos em tarefas complexas de manufatura, automação, etc. Já

na *Logistics League* o objetivo é desenvolver robôs e estratégias que atuem de forma flexível em toda a linha de produção. Os robôs tem como atividades: pegar a matéria-prima do estoque, transportá-lo em uma sequência dinâmica de máquinas, manipular o processo nessas máquinas e por fim, entregá-los. Uma equipe é composta por 3 robôs, onde cada robô tem como base o modelo Festo Robotino onde as equipes podem adicionar sensores no mesmo. Essa liga é a mais nova da competição.

- A *RoboCupJunior* é um projeto de caráter educacional, que tem como participantes crianças e adolescentes que estejam no ensino fundamental e médio. Também é permitido que alunos do ensino superior possam participar, desde que não tenham recursos para participar das outras ligas já citadas. A *RoboCupJunior* propõe vários desafios, servindo como uma espécie de introdução a robótica, desenvolvendo habilidades e conferindo experiência aos participantes em disciplinas como eletrônica, *hardware* e *software*.

1.1.1 *Standard Plataform League*

A *Standard Plataform League* ou SPL é uma categoria onde todas as equipes partem do mesmo ponto, onde não há vantagens em termos de *hardware*, já que todos devem utilizar a mesma plataforma, que não pode sofrer nenhuma modificação por parte dos competidores. Desde o ano de 2009 até o presente, 2016, a plataforma utilizada é o robô NAO. Nesta liga todos os robôs devem ser completamente autônomos em relação a tomada de decisões e os jogadores recebem as informações através do *Game Controller*. A liga possui um conjunto de regras que podem ser conferidas em [1] onde o supervisionamento do cumprimento delas, além de passar a informação aos jogadores sobre informações relativas as penalizações, gols marcados, começar, parar e terminar a partida são tarefas competente ao árbitro, onde o *Game Controller* é o *software* presente em apenas um computador em cada campo, que é operado pelo árbitro e serve como ferramenta de comunicação entre ele e os robôs.

As partidas da SPL, além de possuírem o árbitro na forma do *Game Controller*, são disputadas por duas equipes que devem ter no mínimo seis jogadores: cinco em campo e mais um na reserva. Ao atuarem como um time completo, as equipes possuem um grande número de estratégias possíveis para abordarem, tendo jogadores específicos para a defesa, ataque, mapeamento e localização. Assim, é o conjunto formado por todas as possibilidades possíveis que enriquece a competição.

Dentro da SPL ainda existe uma subcategoria chamada *Drop-in*. Dentro dessa subcategoria, diferentemente da anterior, as equipes mandam apenas um jogador em cada partida. Os times são compostos por cinco equipes diferentes com um jogador de cada equipe e são decididos por sorteio, sem conhecimento prévio dos participantes.

Na *Drop-in* é avaliado o melhor jogador ao longo de várias partidas e não mais o melhor time, já que eles são mutáveis a cada partida. Ao termos cinco equipes diferentes compondo um time sem nenhum conhecimento prévio sobre a escalação dos times, existe uma série de complicações se comparadas com a modalidade principal, como por exemplo: não é possível garantir que seus aliados transmitam informações confiáveis. A priori, não existe uma estratégia combinada, portanto é mais difícil fazer a cooperação com robôs estranhos. Outra complicação é que, uma das estratégias mais utilizadas para a localização durante as partidas na modalidade principal é baseada no uso de no

mínimo dois robôs para se obter diferentes perspectivas do campo, diminuindo assim a complexidade do problema.

Na *Drop-in*, devido as complexidades de comunicação mencionadas, o jogador só pode utilizar leituras próprias para realizar essa tarefa. O desafio inerente ao sistema de localização baseado em câmeras monoculares e um sistema de navegação inercial não confiável é o que motiva a confecção desse trabalho: resolver o problema da localização na modalidade *Drop-in* da SPL.

O campo de futebol da categoria SPL, tem dimensões e posicionamentos previamente conhecidos, assim, as informações do mapa de atuação do robô já são conhecidas de antemão, o que facilita o trabalho da localização, pois podemos pular a etapa de mapeamento e usar estas informações diretamente para tentar extrair informações do ambiente.

1.2 ESTADO DA ARTE

Por ser um problema bem abrangente tanto em ambientes quanto plataformas, uma vez que dada a aplicação, os sensores, o processamento, a coleta e o tratamento destas informações para transformá-las em conhecimento podem variam bastante. Pensando neste cenário, existem vários trabalhos aos quais podemos destacar. Ingemar J. Cox em [2] propõe um veículo (BLANCHE) cujo objetivo é operar de maneira autônoma em um ambiente de chão de fábrica ou escritório, onde o maior problema consiste-se em estimar com precisão a posição (x, y, θ) deste veículo. BLANCHE foi projetado como um veículo de baixo custo e, portanto, em relação a parte sensorial só possuía um sensor óptico de distância e odometria nas rodas. A estimação de posição deste robô era feita em quatro etapas:

1. Inicialmente o robô deveria ter um mapa do ambiente.
2. Era feita uma combinação da odometria e do sensor óptico.
3. Utilizava um algoritmo para casar a informação dos sensores com o mapa obtido previamente e
4. calculava a distância de Mahalanobis para estimar a precisão com que foi feita a junção das informações no passo 3.

Em [3], Leonard e Durrant-Whyte apresentam uma aplicação do filtro de Kalman estendido para solucionar o problema de navegação de um robô em um ambiente previamente conhecido. Neste trabalho, foi feita a tarefa de localização nesse ambiente em duas plataformas: A primeira era um robô que possuía apenas um sonar rotativo e a segunda era um robô Robuter que possuía seis sonares em posições fixas. No ambiente de trabalho foram fixados marcadores que assim determinavam certas posições no mundo. Nesse trabalho, no que correspondia a etapa de predição do FKE foi utilizado um modelo desenvolvido que representava a movimentação do robô ao longo do tempo em resposta a uma entrada e a correção ficava a cargo das observações das marcações espalhadas pelo ambiente.

Trabalhos como estes foram fundamentais para o desenvolvimento dos algoritmos utilizados na *RoboCup*, servindo como base para o desenvolvimento dos times na competição. Portanto já pensando no campo de futebol, com passar dos vários anos de competição, muitas pesquisas de como se

localizar em campo já foram desenvolvidas. No começo essa tarefa era mais simples, uma vez que o gol de cada time era de cores diferentes: um azul e outro amarelo. Como apresentado em [4] por Whelan T., Studli S., McDonald J., Middleton R. H., que por sua vez é baseado no trabalho original feito por Cox [2] onde os autores de [4] já haviam feito uma adaptação no trabalho de Cox, substituindo o sensor óptico por uma câmera de vídeo, podendo ser conferido em [5]. Ele tem como sua maior contribuição justamente adaptar o algoritmo de Cox para o contexto da SPL. Propondo assim o algoritmo modificado de Cox, que basicamente adiciona todas as marcações do campo (traves, linhas, centro de campo), a estimação de posição baseado na distância desses elementos e a aplicação de mínimos quadrados para minimizar o erro de estimação. Nesse trabalho, também é proposto a utilização do diagrama de Voronoi para que se reduza o custo computacional do processo de determinação da distância até as *features*. O algoritmo de Cox modificado pode ser descrito em três etapas diferentes:

1. Dado um conjunto de pontos detectados, primeiramente é feito a projeção do ponto da imagem para o sistema de coordenadas do mundo.

2. Após projetado esses pontos no mundo é feita a seleção de pontos no mundo mais próximas dessa marcação, nessa etapa que é utilizado o diagrama de Voronoi, pois só é necessário processar uma vez no *setup* do robô, assim é possível determinar o ponto em tempo *O(1)*.

3. Por fim, encontra-se a correção para a pose atual, utilizando várias técnicas mostradas no artigo.

Neste segundo trabalho dos autores no ambiente *RoboCup*, a localização é feita a partir da detecção de determinadas features do campo como as linhas e gols, que por serem de cores diferentes, agregavam maior informação ao processo. Podendo assim, baseado na cor, determinar qual lado era qual no campo. Já para fundir as informações da visão e odometria era utilizado o *Unscented Kalman Filter* (UKF). Onde também era feita uma comparação com os resultados obtidos pelo filtro de Kalman e o filtro de Kalman Estendido implementado em trabalhos passados. Onde ao final, foi mostrando que o desempenho do UKF era superior. O interessante nesse trabalho é que eles estabeleceram como métrica de avaliação um erro máximo tolerável para se considerar exata a pose estimada do robô, sendo 30cm da posição correta no eixo xy e 15º para a orientação.

Alguns anos depois, as cores dos gols foram trocadas, sendo ambos na cor amarela. Essa simples modificação aumentou o grau de dificuldade da tarefa em estudo, uma vez que tendo a mesma cor, agrega-se uma certa ambiguidade na detecção do gol, onde utilizar a informação do gol não é mais determinística para identificar de qual lado do campo o robô se encontra. Com estas modificações, em [6], Sánchez *et al*. utilizam, assim como nos demais trabalhos citados, técnicas de visão computacional para detecção de pontos de interesse no campo para que assim se obtenha uma medida de distância destes pontos ao robô. Porém, diferentes dos outros trabalhos que utilizavam filtro de Kalman e seus variantes para estimativa de posição, em [6] eles utilizam filtro de partículas baseados na técnica de Monte Carlo. Neste artigo ainda é apresentada uma técnica de tratamento de imagens para que se possa estimar a distância das features em relação ao robô.

Em [7], Anderson *et al*. propõem uma alternativa diferente dos outros trabalhos até agora, mostrando uma técnica chama de ICP - *Iterative Closest Point*. Este método calcula a posição estimada do robô levando em consideração a posição inicial do robô, o mapa do ambiente e o conjunto de features observadas naquele instante, onde a saída desse método é utilizada como entrada para o filtro de

Kalman para aumentar a precisão da estimativa. Como dito no artigo, este método apresenta alguns problemas, sendo um deles quando todas as observações são extraídas do mesmo frame da câmera e isto ocasiona a deterioração da acurácia do processo. A ideia principal do algoritmo proposto é atribuir valores sobre cada marcação que pode ser identificada no campo (linha, gol, trave, encontro de linhas) e como ela se comporta perto de outras marcações, como por exemplo, ao encontrar uma trave próxima de uma junção de linhas é possível reduzir as possíveis posições que o robô possa estar.

Novamente tentando aproximar o campo da SPL para o campo real, os gols passaram para a cor branca. Atitude essa que novamente aumenta o grau de dificuldade da competição, já que a detecção do mesmo se torna mais difícil pois o mesmo pode ser confundido com as linhas brancas do campo. Até o presente momento não existem trabalhos sobre o campo com esta configuração. Portanto a tarefa de localização na *RoboCup* ainda é um problema intrigante que inspira pesquisas em várias universidades ao redor do mundo.

2 FERRAMENTAS UTILIZADAS

Nesse Capítulo primeiro será introduzido o robô humanoide NAO, citando todos seus sensores e entrará-se em detalhes apenas sobre sua unidade inercial de movimento e as câmeras de vídeo. Após apresentado o NAO e seus sensores, será detalhado as configurações do campo de futebol que é usado na *Standard Plataform League*.

2.1 ROBÔ NAO

A plataforma utilizada neste trabalho, é o robô NAO, desenvolvido pela empresa *Aldebaran Robotics* (Figura 2.1), é um robô humanoide de 58cm de altura, desenvolvido em 2006 para que pudesse fazer companhia ao homem [8].

Este é o modelo adotado na categoria *Standard Plataform League* (SPL) da *RoboCup* desde o ano de 2008. Ele é dotado de uma série de sensores, permitindo assim ser usado nas mais diversas aplicações, sendo possível trabalhar competências nas áreas de visão computacional, interação homem-robô, cooperação multitarefas com outros robôs. Onde nas subseções seguintes serão mostrados os sensores que ele possui e apresentaremos melhor as especificações da câmera e da IMU.

2.1.1 Especificações do NAO

O NAO possui uma série de sensores espalhados por todo o corpo, como pode ser conferido na Figura 2.2.

Segundo especificações do fabricante, ele apresenta:

- 2 Câmeras na Cabeça;
- Sensores táteis na cabeça e nas mãos;
- Microfones;
- Alto-falantes;
- Botão no peito;
- Sensores infravermelho;
- Dois pares de sensores ultrasom no peito.
- Uma unidade inercial (IMU) composta de um acelerômetro e girômetro nos três eixos localizado no torso.
- *Bumper* e sensores de pressão em cada um dos pés.

Figura 2.1: Robô NAO
fonte: Aldebaran Robotics

Para que o robô seja capaz de se localizar utilizando a abordagem adotada nesse trabalho, a IMU e ambas as câmeras são fundamentais para esse processo, por esta razão, estes sensores serão melhor especificados nas subseções a seguir.

2.1.1.1 Câmeras de Vídeo

Como visto na subseção anterior, o NAO é composto por duas câmeras, segundo a documentação oficial do fabricante [9] as câmeras possuem uma resolução máxima de 1280x960 a 30 *frames* por segundo, o que na prática pode ser inviável. Uma vez que o robô roda vários processos como locomoção, comportamento e até mesmo a própria visão que é responsável por capturar as imagens e tratá-las em busca de objetos de interesse, como a bola, gol e linhas. Essa resolução e *framerate* se tornam inviáveis, uma vez que essa grande carga de processamento inviabiliza trabalhar com tamanha qualidade, assim, na prática, a resolução máxima utilizada é de 640x480 a um *framerate* variável.

Ambas as câmeras são localizadas na cabeça do robô. Uma está na parte central superior enquanto a outra na parte central inferior, como pode ser visto na Figura 2.2. Estando em diferentes posições, como esperado, elas fornecem visões diferentes. Contextualizando na partida de futebol por exemplo, a câmera superior é ideal para localizar as features de interesse que se encontram mais distantes do jogador, enquanto a inferior já é utilizada para fazer a identificação em regiões próximas, como por exemplo verificar se a bola está próxima aos pés do robô. A localização exata e o campo de visão (field of view, FOV) de cada câmera são ilustradas nas Figuras 2.3 e 2.4.

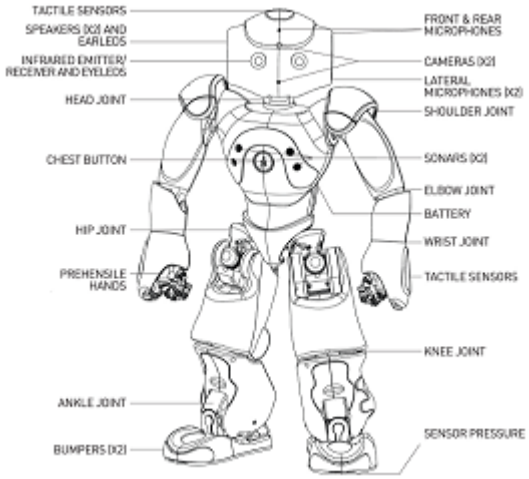


Figura 2.2: Sensores do Robo NAO
fonte:Aldebaran Robotics

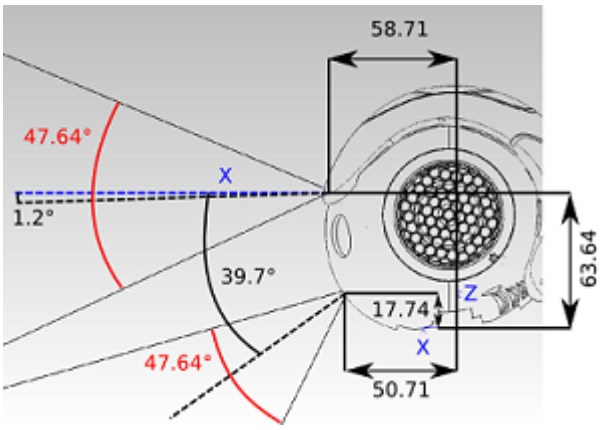


Figura 2.3: Disposição das cameras e fov vertical
fonte:Aldebaran Robotics

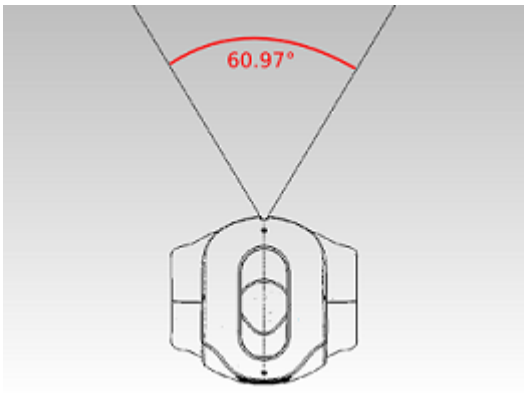


Figura 2.4: Fov Horizontal
fonte:Aldebaran Robotics

2.1.1.2 Unidade de Sensores Inerciais (IMU)

A unidade inercial do NAO é localizada no peito e possui tanto o girômetro quanto o acelerômetro em seus três eixos: X, Y e Z (Figura 2.5).

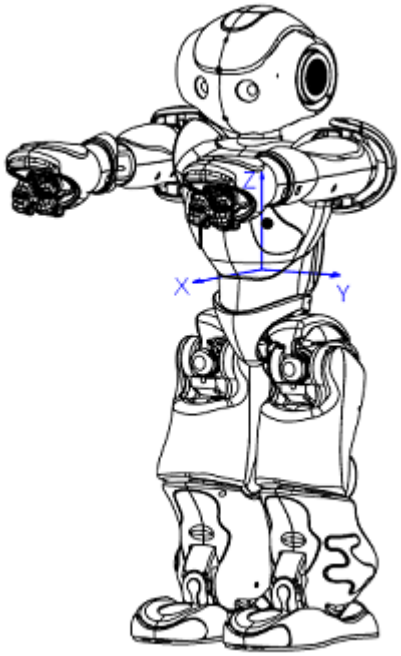


Figura 2.5: Origem do sistema de coordenadas dos eixos X, Y e Z em relação ao robô.
fonte: Aldebaran Robotics

Para determinar a orientação do torso, a Aldebaran implementou um algoritmo que utiliza o acelerômetro para o caso estático, onde somente a aceleração da gravidade pode ser medida. Caso exista movimento, o girômetro é utilizado, porém com a integração do valor do mesmo é gerado um erro associado a medida [10], erro este que será abordado em uma subseção futuramente.

2.2 O CAMPO DE FUTEBOL NA CATEGORIA SPL

Nos primórdios da competição, o campo apresentava algumas configurações diferentes das utilizadas nos dias de hoje. Configurações essas que tornava o ambiente mais simples para a tarefa de localização. No princípio, os gols de cada lado do campo apresentavam cores diferentes, sendo um amarelo e outro azul, onde uma vez detectada a cor do gol poder-se-ia saber em qual lado do campo o robô está posicionado, bem como restringir possíveis orientações. Posteriormente, as cores dos gols foram modificadas, ambos sendo da cor amarela, com essa medida, apenas identificar a cor do gol já não era determinístico para a posição, precisando ser elaborada uma estratégia a mais para obter essa informação. Finalmente em 2015 as cores dos gols foram novamente alteradas, para a cor branca, assim como é no futebol da FIFA. Essas mudanças,apesar de tornarem o ambiente mais realista, dificultaram bastante os processos da competência da visão computacional, uma vez que a detecção do gol e das linhas do campo, ambas nas cores brancas, tornou essa caracterização mais complexa.

A bola de futebol também sofreu alterações ao longo dos anos, onde inicialmente era dar cor laranja e no ano de 2016 passou para o modelo mais tradicional de bola de futebol: branca com hexágonos pretos. Também foi mudado o material da bola. Esse novo material absorve melhor o impacto, podendo assim diminuir a movimentação da bola após chutada. Todas essas modificações ao longo dos anos são pequenos passos para o cumprimento da meta principal no futuro, de se ter robôs humanoides capazes de competir contra humanos sob as regras da FIFA. Até o presente momento, o campo de futebol da categoria SPL bem como as dimensões de cada elemento presente no campo são ilustrados na Figura 2.6 e na Tabela 2.1 e as especificações do gol podem ser encontradas na Figura 2.7.

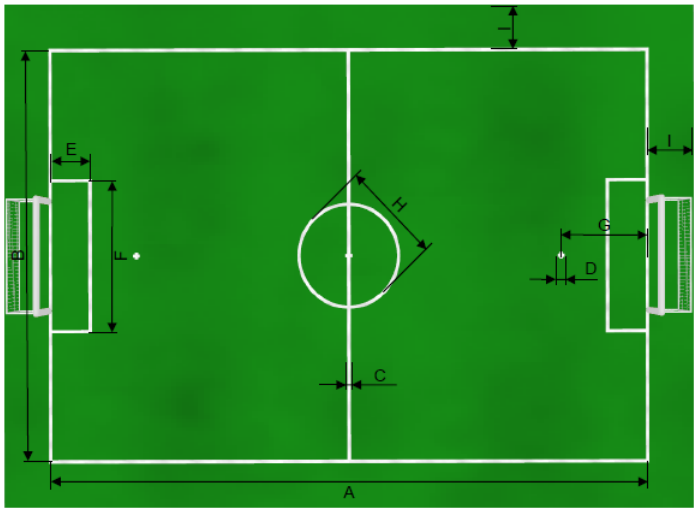


Figura 2.6: Campo de Futebol SPL
fonte:Rules 2016 SPL

Tabela 2.1: Dimensões do Campo

ID	Descrição	Tamanho (mm)
A	Comprimento do campo	9000
B	Largura do campo	6000
C	Largura da Linha	50
D	Tamanho da marca do Pênalti	100
E	Comprimento da Área do Pênalti	600
F	Largura da Área do Pênalti	2200
G	Distância da marca do Pênalti	1300
H	Diâmetro do Centro de Campo	1500
I	Distânciada linha até o limite do campo	700

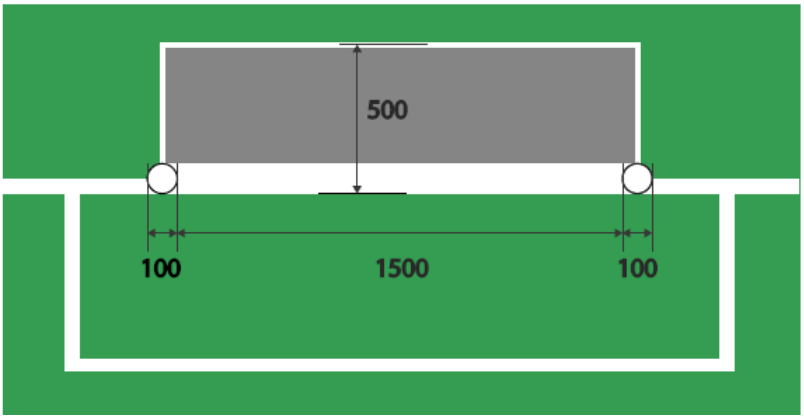


Figura 2.7: Dimensões do Gol
fonte:Rules 2016 SPL

Como pode-se notar, o campo de futebol apresenta todos seus elementos duplicados, ou seja, não existe no campo uma *feature* que possa ser utilizada para demarcar com exatidão uma posição ou orientação. Portanto é necessário trabalhar alguma maneira para retirar a ambiguidade do ambiente e/ou caracterizar as marcas do campo. Estratégias essas que serão abordadas em seções futuras.

3 FUNDAMENTAÇÃO TEÓRICA

Este capítulo busca dar ao leitor uma base matemática e conceitual, para que o mesmo possa melhor acompanhar o desenvolvimento do trabalho. Primeiramente será introduzido o conceito de transformação de coordenadas homogêneas, cobrindo tanto a translação quanto a rotação. Após isso será apresentado o conceito de projeção, quatérnios e por fim, o filtro de Kalman, bem como sua versão estendida, o Filtro de Kalman Estendido (FKE ou EKF do inglês *Extended Kalman Filter*).

3.1 TRANSFORMAÇÃO DE COORDENADAS HOMOGÊNEAS

Um ponto pode ser representado no espaço tridimensional como um vetor contendo sua localização em cada componente dos eixos X, Y e Z

$$p_1 = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}, \tag{3.1}$$

um operador, é qualquer processo que mapeia um ponto em outro ponto, como por exemplo rotação e escala [3], que por sua vez podem ser representados por uma matriz 3x3:

$$p_2 = \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = Op.p_1 = \begin{pmatrix} op_{xx} & op_{xy} & op_{xz} \\ op_{yx} & op_{yy} & op_{yz} \\ op_{zx} & op_{zy} & op_{zz} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}. \tag{3.2}$$

No entanto, o problema de representar operadores da forma 3x3 é que não é possível representar uma translação com um operador com esta dimensão, logo uma operação comum como a translação deve ser representada através de adição de vetores [10].

3.1.1 Transformada Homogêneas

Para que se possa solucionar o problema da translação apresentado, uma solução é projeta-lo em um espaço de quatro dimensões. O quarto elemento representa um fator de escala, onde é possível representar um ponto em quatro dimensões como:

$$p_{1,4D} = \begin{pmatrix} x_1 & y_1 & z_1 & w_1 \end{pmatrix}^T, \tag{3.3}$$

e o mesmo ponto no espaço tridimensional pode ser encontrado dividindo seus elementos pelo fator de escala w:

$$p_{1,3D} = \begin{pmatrix} \frac{x_1}{w_1} & \frac{y_1}{w_1} & \frac{z_1}{w_1} \end{pmatrix}^T, \tag{3.4}$$

onde uma vez que se considere $w_1 = 1$, tem-se o ponto $\boldsymbol{p}_1$ como apresentado em (3.1) podendo então ser agora definido um operador de translação:

$$p_2 = p_1 + p_{trans} = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} + \begin{pmatrix} x_{trans} \\ y_{trans} \\ z_{trans} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & x_{trans} \\ 0 & 1 & 0 & y_{trans} \\ 0 & 0 & 1 & z_{trans} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix}. \tag{3.5}$$

3.1.2 Operadores de Rotação

Uma vez apresentado a translação, pode-se definir os operadores relativos a rotação que pode ser realizada sobre qualquer eixo, x , y ou z e ainda fazer composições de rotação e translação, onde neste caso basta realizar uma multiplicação de operadores, para que se obtenha a posição final do *frame*.

Os operadores de rotação são definidos como:

$$R_{x,\theta} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sen\theta & 0 \\ 0 & sen\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.6}$$

$$R_{y,\theta} = \begin{bmatrix} cos\theta & 0 & sen\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sen\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.7}$$

$$R_{z,\theta} = \begin{bmatrix} cos\theta & -sen\theta & 0 & 0 \\ sen\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.8}$$

sendo $R_{x,\theta}$ a rotação em torno do eixo x em um ângulo θ, $R_{y,\theta}$ a rotação em torno do eixo x em um ângulo θ e $R_{z,\theta}$ a rotação em torno do eixo x em um ângulo θ .

3.2 PROJEÇÃO DA IMAGEM

Para que o robô possa se localizar no espaço, a partir do momento que se utiliza a visão computacional como um dos meios para realizar esta tarefa, saber usar a técnica de projeção pode ser de grande valia. Ao encontrar algum ponto de referência na imagem, este ponto está representando uma coordenada da imagem em pixels (u, v) enquanto informações vindas de outros sensores, podem ser relativas ao robô, como por exemplo a navegação inercial proveniente dos sensores inerciais. Portanto saber transformar um ponto no sistema de coordenadas da imagem para um sistema de coordenadas do robô ou do mundo é fundamental.

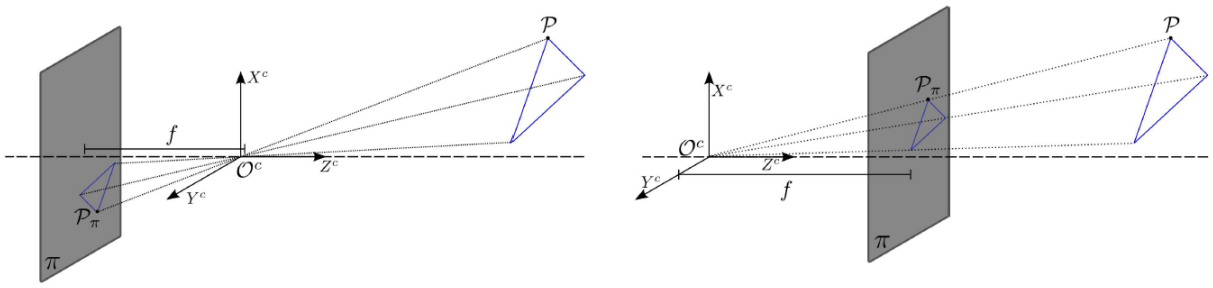


Figura 3.1: Modelo pinhole e modelo pinhole com imagem virtual
fonte: fonte: Bernardes, C. M. [4]

3.2.1 Modelo Pinhole de Projeção Perspectiva

No modelo pinhole pode-se chamar o plano da imagem de plano π sobre o qual as objetos do mundo real são projetados e ao traçar traçar uma linha horizontal é representado o eixo óptico (Z^c) como pode ser visto na Figura 3.1.

A lente da câmera é posicionada perpendicularmente ao eixo óptico na origem do sistema de coordenadas O^c, que é conhecido como centro óptico e o comprimento focal f é um parâmetro intrínseco das lentes e o sistema de coordenadas da imagem é definido por O^i.

Neste modelo apresentado, assume-se que todos os raios de luz captados do ambiente atravessem a lente da câmera em um único ponto, o centro óptico. Os raios de luz são então projetados sobre o plano π, formando uma imagem invertida do objeto. A partir disto, pode ser assumida uma simplificação, fazendo com que a uma imagem virtual seja colocada em um plano em frente do centro óptico a uma distância f [11].

Uma projeção, do ponto de vista matemático, pode ser caracterizada com a ajuda de duas transformações: uma projeção que leva um ponto no espaço tridimensional para o bidimensional projetado sobre o plano da imagem, e uma transformação de coordenadas do sistema O^c para O^i geralmente acrescida de uma conversão de unidades [4].

Dado um ponto p representado no sistema de coordenadas da câmera O^c por p=$(x,\ y,\ z)^T$ é projetado no plano da imagem P_π como pode ser conferido na Figura 3.1, onde por semelhança de triângulos é fácil encontrar as coordenadas correspondentes no plano P_π em O^c.

$$P_\pi = \begin{pmatrix} \frac{fx}{z} \\ \frac{fy}{z} \\ f \end{pmatrix} = f\frac{p}{z}. \tag{3.9}$$

Estabelecida as coordenadas do ponto P_π em O^c, é necessário transformar o sistema de coordenadas O^c para O^i, que pode ser feito com uma rotação e uma translação:

$$R_c^i = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.10}$$

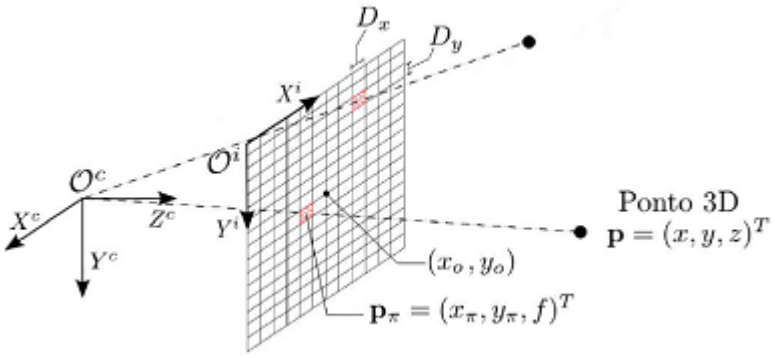


Figura 3.2: Projeção de um Ponto no plano P_π da imagem
fonte: fonte: Bernardes, C. M. [4]

$$t_c^i = \begin{pmatrix} x_o \\ y_o \\ 1 \end{pmatrix}, \tag{3.11}$$

onde x_o e y_o são as coordenadas do centro da imagem medida em pixels e representadas no sistema O^i. Uma representação da transformação de coordenadas é encontrada na Figura 3.2.

Ao deparar-se com dois sistemas com diferentes unidades de medida, um sendo métrico e outro em pixels, é necessário acrescentar uma escala, que pode ser representada por uma matriz diagonal D, onde d_x e d_y representam fatores de escala na horizontal e vertical, respectivamente [4]. Assim pode-se escrever o modelo final para transformação de coordenadas da câmera para coordenadas da imagem como:

$$p^i = \begin{bmatrix} d_x & 0 & 0 \\ 0 & d_y & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} P_\pi + \begin{bmatrix} x_o \\ y_o \\ 1 \end{bmatrix}, \tag{3.12}$$

$$p^i = \begin{bmatrix} -d_x & 0 & x_o \\ 0 & d_y & y_o \\ 0 & 0 & \frac{1}{f} \end{bmatrix} P_\pi = K\frac{p}{z}, \tag{3.13}$$

onde K é a matriz de calibração e pode ser representada como:

$$K = \begin{bmatrix} -fd_x & 0 & x_o \\ 0 & fd_y & y_o \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.14}$$

3.3 QUATÉRNIOS

Os quatérnios foram propostos por Sir William Rowan Hamilton, em 1843, quando o mesmo pesquisava uma maneira de estender o plano complexo ao espaço de três dimensões. Em fevereiro de 1845, A Cayley mostrou em sua pesquisa que os quatérnios também poderiam ser usados para representar orientação [12].

De forma análoga aos números complexos, que podem descrever uma rotação em um plano, os quatérnios são uma estrutura algébrica que descreve rotações no espaço. Comparando-se com outras modelagens, como por exemplo as matrizes de rotação e os ângulos de Euler, os quatérnios são mais estáveis numericamente e mais eficientes por não apresentarem singularidades. Além disso, os quatérnios permitem nos aproximar de soluções de sistemas algébricos integrando parâmetros de rotação desconhecidos [8].

3.3.1 Fundamentação Matemática

Um quatérnio pode ser definido como:

$$q = q_r + q_x\bar{i} + q_y\bar{j} + q_z\bar{k} = q_r + \bar{v}, \tag{3.15}$$

onde $q_r,\ q_x,\ q_y\ e\ q_z$ são escalares, $\bar{v}$ é um vetor e $\bar{i},\ \bar{j},\ \bar{k}$ são três vetores unitários ortogonais de um sistema de coordenadas (x, y, z).

O conjugado é definido como:

$$q* = q_r - q_x\bar{i} - q_y\bar{j} - q_z\bar{k} = q_r - \bar{v}. \tag{3.16}$$

Um quatérnio composto por apenas parte vetorial, pode representar uma posição no espaço 3D da seguinte maneira:

$$\bar{r} = q_x\bar{i} + q_y\bar{j} + q_z\bar{k}. \tag{3.17}$$

Pode-se definir a norma de um quatérnio unitário igual a $\|q\| = q_r^2 + q_x^2 + q_y^2 + q_z^2$= 1 que também pode ser escrito da seguinte maneira:

$$\|q\| = cos\left(\frac{\theta}{2}\right) + sen\left(\frac{\theta}{2}\right)\bar{d}, \tag{3.18}$$

onde $\bar{d}$ é a direção do eixo de rotação, θ é o angulo de rotação, e este quatérnio é chamado de Quatérnio de Rotação [5].

3.3.2 Quatérnio de Rotação

Como apresentado na subseção anterior, em (3.12) definimos o que é um quatérnio de rotação, agora será apresentado como pode-se realizar a operação de rotação utilizando quatérnios.

Usando um vetor $\bar{p}$, ao rotacioná-lo utilizando um quatérnio unitário q, esse vetor pode ser reescrito como um quatérnio puro, uma vez que sua parte real é nula, $p = (0, \bar{p})$.

Assim:

$$r_{rot}(p) = qpq^{-1}, \tag{3.19}$$

onde pela definição da inversa de quatérnios, que pode ser escrita como:

$$q^{-1} = \frac{q^*}{\|q\|^2}, \tag{3.20}$$

onde pode-se verificar facilmente que $q^{-1}q = qq^{-1} = 1$ e caso q seja um quatérnio unitário [1] obtem-se:

$$q^{-1} = q^*, \tag{3.21}$$

logo, pode-se extender a definição do $q_{rot}(p)$ para:

$$r_{rot}(p) = qpq^*. \tag{3.22}$$

Caso deseja-se aplicar duas ou mais rotações em um corpo, pode-se utilizar a composição de quatérnios. Suponha-se que existam duas rotações distintas, representadas por q_1^0 e q_2^1 sobre o ponto p tem-se:

$$q_2^1(q_1^0(p)) \ = q_2^1\left(q_1^0 p q_1^{0*}\right) = q_2^1 q_1^0 p q q_1^{0*} q_2^{1*}, \tag{3.23}$$

fazendo $q_2^0 = (q_2^1 q_1^0)$ e $q_2^{0*} = (q_2^1 q_1^0)*$, pode-se representar a composição das rotações q_1^0 e q_2^1 diretamente por q_2^0[8].

3.4 FILTRO DE KALMAN

O filtro de Kalman foi proposto em 1950 por Rudolph Emil Kalman, como uma técnica de filtragem e predição para sistemas lineares. Em sua versão clássica, ele só pode ser implementado para problemas que possuem estados contínuos, ou seja, ele não é válido para casos discretos ou híbridos [13].

A grande vantagem do filtro de Kalman é que ele é um algoritmo recursivo ótimo em relação a minimização do erro e é composto por um conjunto de equações que provê uma medida estimada eficiente do estado de um processo, baseado no estado anterior, ou o inicial, e levando algumas medidas junto a alguma entrada de controle. Pode-se dizer que o princípio do funcionamento do filtro é que ele prediz um estado de um processo baseado no estado anterior e levando em consideração um certo controle e em uma próxima etapa, ele corrige a predição baseada na medida realizada pelo sistema [10].

O filtro de Kalman é composto por algumas equações, em que a partir de um processo estocástico que seja da forma:

$$x_k = Ax_{k-1} + Bu_k + w_{k-1} \quad , \tag{3.24}$$

e uma medida que seja:

$$z_k = Hx_k + v_k \quad , \tag{3.25}$$

em que x_{k-1} é o estado do sistema no instante k-1, u_k é um vetor de controle no instante k, A e B são matrizes, em que A é de dimensão n x n onde n é a dimensão do vetor de estados x. B é de tamanho n x m, em que m é a dimensão do vetor de controle u, H por sua vez apresenta um tamanho de m x n e por fim w_k e v_k são ruídos associados.

Esses ruídos w_k e v_k são independentes e são de distribuição gaussiana de média nula.

$$p(w) \ = N(0, \ Q), \tag{3.26}$$

$$p(v) \ = N(0, \ R), \tag{3.27}$$

em que Q e R são matrizes de covariância de ruído do processo (n x n) e da medição (m x m) respectivamente.

3.4.1 O algorítimo do filtro de Kalman

O filtro de Kalman funciona com duas etapas distintas, uma de predição e outra de correção, que também podem ser chamadas de estimativa a *priori* e a outra de *posteriori*.

A predição é composta por duas equações, que podem ser vistas a seguir:

$$\hat{x}_k^- = A\hat{x}_{k-1}^+ + Bu_k, \tag{3.28}$$

$$P_k^- = AP_{k-}^+A^T + Q, \tag{3.29}$$

onde calcula-se a estimativa a *priori* baseada apenas no estado passado $\hat{x}_{k-1}$ e a entrada de controle u_{k-1} que é opcional. Pode-se notar também que a incerteza do sistema aumenta baseada na incerteza no instante anterior e no ruído representado pela matriz de covariância Q. Quanto mais predições ocorrerem sem a etapa da correção, maior ficará a incerteza da estimativa [10].

Uma vez apresentada a etapa de predição, vamos definir as equações relativas a correção:

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1}, \tag{3.30}$$

$$\hat{x}_k^+ = \hat{x}_k^- + K_k(y_k - H_k x_k^-), \tag{3.31}$$

$$P_k^+ = (I - K_k H_k) P_k^- (I - K_k H_k)^T + K_k R_k K_k^T, \tag{3.32}$$

onde K_k é a matriz de ganho que possui dimensão n x m e tem como função minimizar a covariância a *posteriori* P_k. $\hat{x}_k^+$ é a estimativa a *posteriori* que leva em consideração a sua estimativa a *priori*, o ganho do sistema K_k e a medida de predição $H_k x_k^-$ e a leitura no instante k, y_k.

3.5 FILTRO DE KALMAN ESTENDIDO

Como já dito, o filtro de Kalman só é válido para sistemas lineares, o que tornava o uso do filtro restrita a algumas aplicações. Até mesmo neste trabalho não seria possível utilizar o filtro de Kalman como método de fusão sensorial para monitorar a movimentação do robô no campo de futebol, uma vez que sua rotação não pode ser descrita de forma linear.

Para resolver problemas onde o caso linear não se aplica, foi proposto o Filtro de Kalman Estendido (FKE ou EKF, que vem do inglês *Extended Kalman Filter*). Agora assume-se que o processo e a medição são governados por funções não-lineares, sendo elas *f* e *h* respectivamente:

$$x_k = f_{k-1}(x_{k-1}, u_{k-1}) + w_{k-1}, \tag{3.33}$$

$$y_k = h_k(x_k) + v_k, \tag{3.34}$$

3.5.1 O algoritmo do Filtro de Kalman estendido

Assim como no filtro de Kalman tradicional, também apresentam a etapa de predição e correção, onde a predição pode ser definida como:

$$x_k^- = f_{k-1}(x_{k-1}^+, u_{k-1}), \tag{3.35}$$

$$P_k^- = F_{k-1} P_{k-1}^+ F_{k-1}^T + Q_{k-1} W_k^T, \tag{3.36}$$

onde F_{k-1}é a jacobiana que contém derivadas parciais de *f* relativa a x:

$$F_{k-1[i,,j]} = \frac{\partial f_{k-1}[i]}{\partial x[j]} \left(\hat{x}_{k-1}, u_{k-1}\right), \tag{3.37}$$

e podemos representar a etapa de correção como:

$$K_k = P_k^- H_k^T \left(H_k P_k^- H_k^T + R_k\right)^{-1}, \tag{3.38}$$

$$\hat{x}_k^+ = \hat{x}_k^- + K_k \left[y_k - h_k\left(\hat{x}_k^-\right)\right], \tag{3.39}$$

$$P_k^+ = (I - K_k H_k)\, P_k^- \, (I - K_k H_k)^T + K_k R_k K_k^T, \tag{3.40}$$

onde H_ké a jacobiana que contém derivadas parciais de *h* relativa a x:

$$H_{k[i,,j]} = \frac{\partial h[i]}{\partial x[j]} \left(\hat{x}_k^-, u_{k-1} \right). \tag{3.41}$$

Apesar de abordar o caso não-linear, o FKE não é ótimo como o filtro de Kalman tradicional, isso se deve a linearização que é realizada, uma vez que a mesma não apresenta um resultado exato, e sim aproximado. Ainda devido a linearização do processo nossa matriz de covariância não representa a covariância real da estimativa e costuma ser menor do que o valor real de covariância [13].

3.6 NAVEGAÇÃO INERCIAL

Como já apresentado, o robô NAO possui uma IMU composta por acelerômetro e girômetro em seus três eixos, sensores esses que são usados para medir a força específica e a velocidade angular, respectivamente, do robô em cada um de seus eixos. A navegação inercial é um método que utiliza estes sensores inerciais de modo que se obtenha o valor da posição atual.

As seções a seguir explicarão como pode ser entendido a saída de cada um dos sensores e como tratar estes dados para que se obtenha a informação desejada, onde para tal será utilizado a álgebra de quatérnios.

3.6.1 Acelerômetro

O acelerômetro é capaz de medir a aceleração inercial de um corpo junto a aceleração da gravidade do local, portanto é importante que se leve este fator g em compensação a medida realizada pelo sensor, portanto a aceleração inercial pode ser calculada segundo (3.42):

$$a = f + g, \tag{3.42}$$

onde *a* é a aceleração inercial, f o valor medido pelo sensor e g é a aceleração da gravidade.

Uma vez conhecido o valor de sua aceleração inercial em relação a um sistema de coordenada local (l) , é interessante que se obtenha este valor em relação a um sistema de coordenadas global (g), então, para que se realize esta tarefa, uma matriz de rotação $C_{\mathbf{g}}^{\mathbf{l}}$ é estabelecida que represente a rotação de l para g.

Assim, pode-se dizer que a aceleração inercial do robô em relação à referência $a^g = [a_x^g \; a_y^g \; a_z^g]^T$, é dada por:

$$a^g = C_{\mathbf{g}}^{\mathbf{l}} a^l, \tag{3.43}$$

substituindo (3.43) em (3.42), tem-se:

$$a^g = C_{\mathbf{g}}^{\mathbf{l}} f^{\mathbf{l}} + g^g, \tag{3.44}$$

onde g^gé o campo gravitacional medido em g e $f^{\mathbf{l}}$é a força específica medida pelo acelerômetro.

Logo, para que se possa determinar a orientação do robô no mundo, é necessário definir a matriz de rotação $C_{\mathbf{g}}^{\mathbf{l}}$, é nesse ponto que se faz necessário outro sensor, o girômetro.

3.6.2 girômetro

Com o girômetro é possível medir a velocidade angular do robô em seus três eixos, onde uma vez integrando-se este valor, é possível obter a posição angular do mesmo em relação ao seu sistema inercial (Figura 2.7).

Assim, utilizando os valores lidos por estes sensor, temos os valores da velocidade angular ω_x, ω_y, ω_z em torno dos eixos $X^{\mathbf{l}}$, $Y^{\mathbf{l}}$, $Z^{\mathbf{l}}$respectivamente. Utilizando quatérnios unitários, pode-se descrever a orientação de um corpo como sendo $q_{\mathbf{g}}^{\mathbf{l}} = [q_0\ q_1\ q_2\ q_3]^T$ com estado inicial $[1\,0\,0\,0]^T$ e definir a rotação do corpo como:

$$\dot{q}_{\mathbf{g}}^{\mathbf{l}} = -\frac{1}{2}\begin{bmatrix} 0 & \omega_x & \omega_y & \omega_z \\ -\omega_x & 0 & -\omega_z & \omega_y \\ -\omega_y & -\omega_z & 0 & -\omega_x \\ -\omega_z & -\omega_y & \omega_x & 0 \end{bmatrix} q_{\mathbf{g}}^{\mathbf{l}} + e, \tag{3.45}$$

onde e é o erro associado e ruídos dos sensores e:

$$q_{\mathbf{g}}^{\mathbf{l}}(k) \ = q_{\mathbf{g}}^{\mathbf{l}}(k-1) + t.q(k), \tag{3.46}$$

sendo t o período de amostragem, e por último podemos definir a matriz de rotação $C_{\mathbf{g}}^{\mathbf{l}}(q_{\mathbf{g}}^{\mathbf{l}})$:

$$C_{\mathbf{g}}^{\mathbf{l}}(q_{\mathbf{g}}^{\mathbf{l}}) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}. \tag{3.47}$$

3.6.3 Integração da aceleração para obtenção de velocidade e posição

Uma vez definida a matriz de rotação $C_{\mathbf{g}}^{\mathbf{l}}$ finalmente pode-se obter os valores da posição do robô em relação ao sistema de coordenadas do mundo, onde a partir da equação (3.44) podemos calcular a velocidade $v^{\mathbf{g}}$:

$$v^{\mathbf{g}}(k) = v^{\mathbf{g}}(k-1) + t.a(k), \tag{3.48}$$

e por fim a posição $p^{\mathbf{g}}$:

$$p^{\mathbf{g}}(k) = p^{\mathbf{g}}(k-1) + t.v^{\mathbf{g}}(k), \tag{3.49}$$

em que, pode-se expandir os vetores $v^{\mathbf{g}}(k)$ e $p^{\mathbf{g}}(k)$ e escrever de forma mais detalhada:

$$v^{\mathbf{g}}(k) = \begin{bmatrix} v_x^{\mathbf{g}}(k) \\ v_y^{g}(k) \\ \omega_z^{\mathbf{g}}(k) \end{bmatrix} = \begin{bmatrix} v_x^{\mathbf{g}}(k-1) + t.a_x^{\mathbf{g}} \\ v_y^{g}(k-1) + t.a_y^{\mathbf{g}} \\ \omega_z^{\mathbf{g}}(k) \end{bmatrix}, \tag{3.50}$$

$$p^{\mathbf{g}}(k) = \begin{bmatrix} p_x^{\mathbf{g}}(k) \\ p_y^{g}(k) \\ \theta_z^{\mathbf{g}}(k) \end{bmatrix} = \begin{bmatrix} px(k-1) + t.v_x^{\mathbf{g}} \\ p_y^{g}(k-1) + t.v_y^{\mathbf{g}} \\ \theta_z^{g}(k-1) + t.\omega_z^{\mathbf{g}}(k) \end{bmatrix}. \tag{3.51}$$

4 METODOLOGIA

Neste Capítulo será abordado as diversas etapas que compõem esse trabalho, onde primeiro será explicado quais são os pontos críticos e como foi feito para contornar e/ou solucionar estes problemas, em seguida apresentado um simulador desenvolvido para representar o NAO no campo de futebol abordando suas funções implementadas neste trabalho e por fim, o algoritmo de localização desenvolvido.

4.1 LOCALIZAÇÃO NO AMBIENTE DA *ROBOCUP* E ABORDAGEM INICIAL

4.1.1 Descrição do problema

Todo o processo de localização no campo de futebol na categoria SPL apresenta algum tipo de complicação, tanto do ponto de vista do robô NAO, quanto do próprio ambiente em si. No que diz respeito ao robô, ao se tratar de um modelo humanoide, na maioria dos casos o escorregamento ao se deslocar é um problema real. Estes escorregamentos causam erros nas medições que serão propagados para navegação inercial, ocasionando então leituras erradas e consequentemente uma observação errada do posicionamento do NAO. Ainda relativo a navegação inercial, os próprios sensores inerciais não são muito precisos, o que dificulta mais ainda uma navegação inercial com níveis aceitáveis de precisão. Na literatura, existem diversas técnicas para contornar estes problemas, uma vez que o escorregamento e sensores de baixa qualidade são problemas comuns no mundo da robótica. Levando em consideração todos os sensores presentes no NAO bem como o próprio ambiente da *RoboCup*, sem dúvidas a melhor solução para complementar as leituras da navegação inercial e melhorar a estimativa do posicionamento no campo de futebol é utilizar as câmeras do robô.

Trabalhar com a câmera e consequentemente usar técnicas de visão computacional para extrair informações relevantes do ambiente é o caminho lógico a se seguir. Uma vez que temos várias marcações ao longo do campo e teoricamente podem ser detectáveis, assunto esse que será abordado posteriormente. Porém, trabalhar com visão computacional é custoso e nossa plataforma possui um processador bem limitado, o que torna inviável o processamento de imagens em resoluções e *framerate* elevados, prejudicando assim a detecção de elementos importantes na imagem, o que dificulta o processo de estimação da posição no campo.

Ainda no contexto da visão computacional, o campo, como já dito anteriormente, foi passando por alterações ao longo dos anos, tornando-o mais próximo de um campo de futebol real. Na *RoboCup* 2016 a categoria SPL já se contava com as traves do gol, linhas do campo e bola, nas cores brancas, onde é bem complicado realizar a detecção das traves do gol de maneira que ela não seja caracterizada como uma linha. Outro aspecto complexo que o campo apresenta, é que se levantarmos as *features* presentes em campo: gols, linhas, interseção de linhas e centro de campo, todas, com exceção da última, não são elementos únicos e o campo ainda apresenta simetria. Tanto na horizontal, quanto na vertical, tornando muito difícil o processo de caracterização destes elementos. Onde em um primeiro

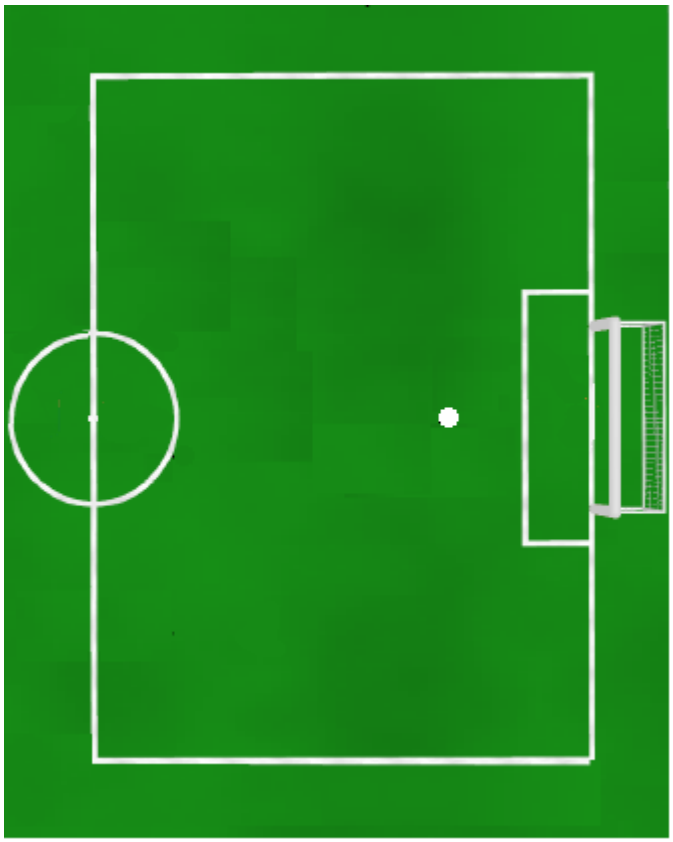


Figura 4.1: Área de atuação do robô referente a meio campo de futebol

momento, apenas encontrando determinadas marcações, na maioria das vezes não será determinístico para a estimação da posição.

4.1.2 Estratégia abordada

Inicialmente, limitou-se o ambiente de trabalho a apenas meio campo, atitude essa que contorna grande parte dos problemas de ambiguidades no campo, reduzindo o número de posições que o robô pode estar ao avistar determinado conjunto de *features*. Essa abordagem é importante, pois reduz a complexidade do problema, facilitando a elaboração de um algoritmo para que se valide neste cenário mais simples e depois partir para aplicação no campo completo que é mais complexo.

Uma vez estabelecida essa abordagem, para que se possa estimar a posição do robô, definiu-se que seria interessante a detecção dos seguintes elementos (*features*):

- Centro de campo;
- Linhas;
- Interseção de linhas;
- Trave do gol;

Este trabalho tem como objetivo maior, elaborar um algoritmo de localização recebendo informações das leituras da unidade inercial (IMU) do NAO advindas do módulo navegação inercial, juntamente

com a detecção e estimativa da distância das *features* até o robô que é provida pelo módulo Visão Computacional. O módulo da visão é proveniente do trabalho que está sendo desenvolvido pela equipe UnBeatables da Universidade de Brasília, onde o método de localização proposto apenas utilizará as informações de distância providas por ele, mais detalhes dos módulos que compõem o NAO será abordado em outra Seção. Neste trabalho também foi desenvolvido um simulador 2D para que se pudesse ter uma maior liberdade para validar o método. Outra contribuição do simulador é a possibilidade de se trabalhar a localização independente da qualidade dos módulos aos quais a localização depende, já que pode-se simular tanto as leituras da navegação inercial, quanto as informações que seriam providas pelo módulo de Visão Computacional, que foi justamente o caso, pois durante o desenvolvimento deste trabalho o módulo de visão não estava sendo capaz de identificar o gol com uma precisão aceitável e sem a ajuda do simulador, a falta desta informação poderia comprometer a validação da eficiência do método proposto. Mais detalhes do simulador podem ser conferidos na próxima Seção.

4.2 SIMULADOR 2D

Para o desenvolvimento do simulador, utilizou-se o *software* MATLAB, já que ele nele é possível implementar equações matemáticas com algumas facilidades, como por exemplo dispensar o uso de pacotes adicionais para matemática, quando comparado com C, Python e outras linguagens.

Estando interessado na pose do NAO (x, y, θ), uma simulação em 3D para este propósito não é necessária, já que o robô sempre estará no chão, bem como todos os elementos do campo também tocam o solo, podemos então assumir que a altura de qualquer objeto é zero (Z = 0), o que facilita muito todo esse processo.

No simulador temos 2 objetos:

1. O campo;
2. O NAO;

4.2.1 O campo

Baseado nas medidas do campo fornecidas pela *RoboCup*, criou-se metade do campo, seguindo todas as especificações ilustradas na Tabela 2.1, possuindo então as quatro linhas que delimitam o campo, o centro de campo, área e traves do gol, tendo como origem do sistema de coordenadas em relação ao mundo o centro de campo. Este campo é um vetor composto por pontos que possuem a coordenada global x,y do ponto no mundo.

4.2.2 O NAO

Para que fosse possível simular a pose do robô no mundo seria interessante que o simulador apresentasse as seguintes funcionalidades:

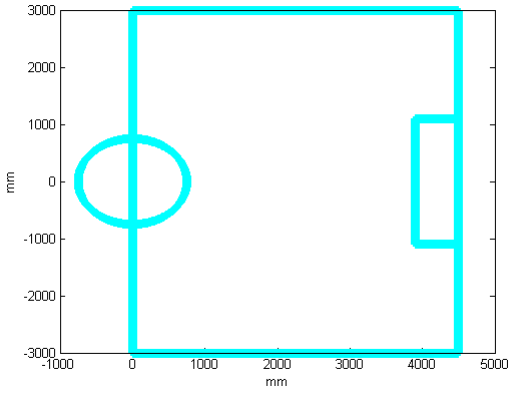


Figura 4.2: Meio campo implementado no simulador

- Posicionar o robô em qualquer local do campo;
  - Ser possível definir as coordenadas x,y e a orientação θ que o robô está atualmente;
- Ser capaz de se mover pelo campo;
  - Atualmente existe 4 possibilidades de movimento que são chamadas aleatoriamente:
    - Deslocamento para uma posição $x_{k+1} > x_k$ e $y_{k+1} > y_k$, com uma variação pequena de θ;
    - Deslocamento para uma posição$x_{k+1} < x_k$ e $y_{k+1} < y_k$, com uma variação pequena de θ;
    - Rotacionar o robô para uma orientação de aproximadamente +30º da orientação no instante anterior;
    - Rotacionar o robô para uma orientação de aproximadamente -30º da orientação do instante anterior;
- Possuir uma pose (x,y,θ) de sua posição simulada verdadeira e sua posição "lida" pela navegação inercial simulada, além da velocidade simulada verdadeira e a lida pela navegação inercial que o NAO apresentou entre um instante e outro;
  - Em cada função de movimento é atribuído a posição verdadeira no campo simulado, o deslocamento efetivo simulado que foi realizado entre um instante e outro, obtendo assim a posição atual, enquanto na posição lida pela navegação inercial além do deslocamento efetivo é adicionado um ruído, ocasionando assim uma leitura diferente do "real", algo próximo de como funciona na prática, sempre há um erro associado a uma medida;
- Em cada posição, poder "ver" as mesmas coisas que ele deveria ver se estivesse nessa posição no campo do mundo real;
  - Baseado no FOV horizontal (sigla que vem do inglês *field of view*) da câmera, informação esta que é informada pelo fabricante, e da distância máxima que é possível identificar algum objeto, podemos calcular qual seria o campo de visão do robô naquele momento, que pode ser aproximado por um triângulo.

- Ser capaz de identificar os objetos de interesse e saber a distância relativa da *feature* até o robô de cada objeto avistado no *frame* por ele;

  - Se tratando de uma simulação e como já é conhecido o mapa, identificar algum objeto nesta simulação, para algumas *features* como interseção de linhas, traves do gol e centro de campo, se resume a saber se determinados pontos do mundo estão ou não dentro do campo de visão do NAO. Já para as linhas foi realizado outro procedimento para identificar se aqueles pontos que estavam dentro do campo de visão e não foram caracterizados como os outros elementos, poderiam formar uma linha, que basicamente foi verificar se os pontos adjacentes mantinham a mesma coordenada no eixo x ou y.

Ou seja, o robô simulado é capaz gerar todas as entradas necessárias para a implementação do nosso algorítmo de localização.

4.2.2.1 Simulação do campo de visão e detecção dos elementos do campo

Para a implementação da simulação da visão do robô, foi necessário saber o campo de visão (propriedade conhecida como fov que vem do inglês field of view) das câmeras que o NAO possui, essa informação é provida pelo fabricante e é mostrada nas Figuras 2.3 e 2.4. Para a completa detecção pelo simulador dos elementos do campo, assim como o robô real faria, foi analisado o trabalho implementado do módulo de Visão Computacional desenvolvido pela equipe UnBeatables em busca da maior distância que o NAO consegue realizar a detecção de alguma feature na resolução máxima de 620 x 480, e foi constatado que é cinco metros. Com base nessas duas informações: Distância de detecção e ângulo de visão, é possível aproximar o campo de visão do robô por um triangulo que simula os elementos avistados. Onde ainda foi adicionado um ruído artificial que representa o ruído associado as câmeras reais, ruídos reais esses que não foram levantados.

Por estar trabalhando em uma simulação, a partir das métricas conhecidas do campo de futebol da SPL é possível caracterizar alguns elementos do campo a partir da seguinte metodologia:

1. Calcula-se quais pontos do campo estão contidos no triangulo que representa o campo de visão do robo;

2. Pelo conhecimento prévio do mapa, já se sabe em quais coordenadas os elementos de interesse estão, portanto para cada ponto:

   (a) Consulta-se se o ponto em questão representa uma coordenada que corresponde a posição de uma quina;

   (b) Consulta-se se o ponto em questão representa uma coordenada que corresponde a posição de uma trave do gol;

3. Caso a resposta seja negativa para essas comparações, verifica-se se existe um subconjunto de pontos que representem pelo menos 1/3 do círculo que compõe o meio de campo, caracterizando assim a detecção do centro de campo;

4. Se todas as outras verificações falharem, tempos que esse ponto é uma linha e para verificar o início e fim da mesma, é feita um procedimento que analisa todo o vetor e para o início e fim de uma linha é acompanhado se há um crescimento sequencial entre os pontos com a manutenção de um dos eixo constantes, por exemplo: os pontos (0;3000), (1;3000), ..., (4500,3000) ou qualquer subconjunto desta sequência representa uma linha, pois há a alteração sequencial de uma coordenada enquanto a outra permanece fixa (y = 3000).

5. Após conhecida o tipo de feature que o ponto pertence, é calculada a distância do ponto em relação ao robô em respeito ao sistema de coordenadas do mundo, sendo essa informação de distância em relação a um determinado tipo de elemento do campo que deveria ser fornecida pelo módulo da Visão Computacional e é a informação que será usada de entrada para o algoritmo de localização.

4.2.2.2 Navegação inercial simulada

Para simular a navegação inercial, foi feito um experimento usando o robô real com o objetivo de coletar os dados dos sensores inerciais do mesmo. Onde foi feita uma coleta dos dados provenientes dos sensores inerciais com o robô parado por cerca de 10 minutos e calculado o erro medio bem como a covariância.

Para o calculo da movimentação, foram implementadas (3.50) e (3.51) e adicionado um ruído branco com os valores calculados através do experimento real.

4.3 A ESTRATÉGIA DE LOCALIZAÇÃO

A motivação principal deste trabalho, é desenvolver um algoritmo de localização, que possibilite o robô saiba se localizar no mundo. Para simplificar o problema e as inúmeras ambiguidades presentes no ambiente, resolveu-se abordar este problema levando em consideração apenas meio campo.

Partindo desse ponto e levando em consideração o robô NAO, como já dito nas seções anteriores, temos duas maneiras distintas de determinar a posição do robô em campo, são elas:

1. Navegação inercial;

2. Câmera monocular;

A navegação inercial, sabemos não ser confiável, não só pelos erros associados a medições bem como por haver problemas de escorregamento e queda do robô, o que reduz ainda mais a confiabilidade das leituras. Todas estas incertezas associadas não permitem que se faça uma estimativa com um nível de confiança aceitável. Já o sistema de visão computacional puro também é incapaz de estimar com precisão a maior parte do tempo a posição do robô, pois nem sempre é possível identificar no *frame* um conjunto de elementos que representem uma única posição provável. Portanto podemos ver que com apenas essas duas diferentes formas de estimação, não seria possível se localizar com precisão. Porém o que acontece se juntássemos estas duas informações?

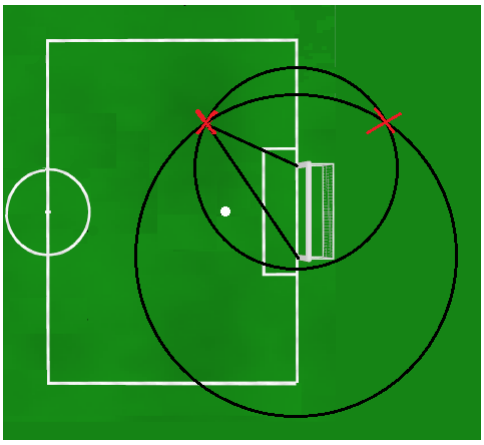


Figura 4.3: Posições prováveis dada a observação de duas traves do gol e conhecendo sua distância até elas.

A fusão sensorial, é uma técnica bem difundida na robótica, onde, a partir de duas ou mais medições, ao combina-las, sempre é possível obter uma informação mais precisa. No nosso caso, podemos combinar nossas leituras utilizando o filtro de Kalman estendido, precisamos implementar a versão estendida do filtro uma vez que lidamos com uma não-linearidade no caso da rotação, como será justificado mais a frente, caso em que a versão tradicional do mesmo é incapaz de cobrir.

Como descrito na Seção 3.5, no FKE temos duas etapas principais: A predição e a correção. Ao caracterizarmos a predição como nossa navegação inercial, temos o seguinte conjunto de equações que descreve a movimentação do robô ao longo do tempo:

$$f(\hat{x}_{k-1}, u_k) = \begin{bmatrix} x_{k-1} + T.v_x[k] \\ y_{k-1} + T.v_y[k] \\ \theta_{k-1} + T.\omega[k] \end{bmatrix}, \tag{4.1}$$

onde x_{k-1} é a posição do robô no eixo x no instante anterior, y_{k-1} é a posição do robô no eixo y no instante anterior, T é o período de amostragem, v_x é a componente da velocidade em x, v_y é a componente da velocidade em y, θ é a orientação do robô e ω é a velocidade angular.

Durante a leitura da navegação inercial, é importante ressaltar que as leituras são feitas em relação ao robô, ou seja, seu zero é a posição inicial a partir da qual o robô começou a se deslocar, na categoria SPL existem cinco posições prováveis que o robô pode começar, posição essa que é escolhida pelo próprio time, assim é possível saber, aproximadamente qual a posição inicial do mesmo, adicionando-a a leitura da navegação inercial.

Já para a etapa de correção, precisamos definir a função não-linear h_k que também nos forneça uma estimativa de posição e orientação. Do sistema de visão computacional, é possível obter qual a distância do robô até os elementos avistados através dá câmera, onde, como já conhecemos o mapa do ambiente, ao sabermos a distância em x e y do robô até a *feature* e conhecendo suas possíveis posições globais no mapa, pode-se criar um conjunto de regiões prováveis. O melhor caso para exemplificar esse procedimento é quando é possível avistar as duas traves do gol, como ilustra a Figura 4.3.

Onde avistar as duas traves do gol e conhecendo suas distâncias até cada trave, constituí em uma

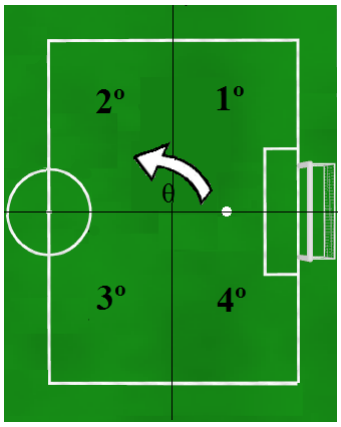


Figura 4.4: Sentido da orientação em relação ao sistema de coordenadas do mundo.

posição única, já que uma das duas opções está fora dos limites do campo. Em contrapartida, o pior caso seria aquela ao qual é possível avistar apenas uma quina, tendo então N prováveis posições já que a quina é o elemento mais comum nesse mapa. Uma vez estabelecido um método para a estimação da posição x,y do robô no campo, é necessário criar um procedimento para a estimação da orientação θ do robô em relação ao mundo. Esse processo é ainda mais complexo que o anterior. Onde a orientação do robô no campo é descrita na Figura 4.4.

Para que se possa estimar o valor da orientação do robô em relação ao mundo através da visão computacional, desenvolveu-se o seguinte método:

1. Primeiramente mudamos o sistema de coordenadas, considerando o robô como origem (0,0) mas conservamos sua orientação, que ainda é desconhecida;

2. Nesse novo sistema de coordenadas, a posição da *feature* será justamente a distância dela até o robô em relação ao mundo, que já foi fornecida pela visão simulada;

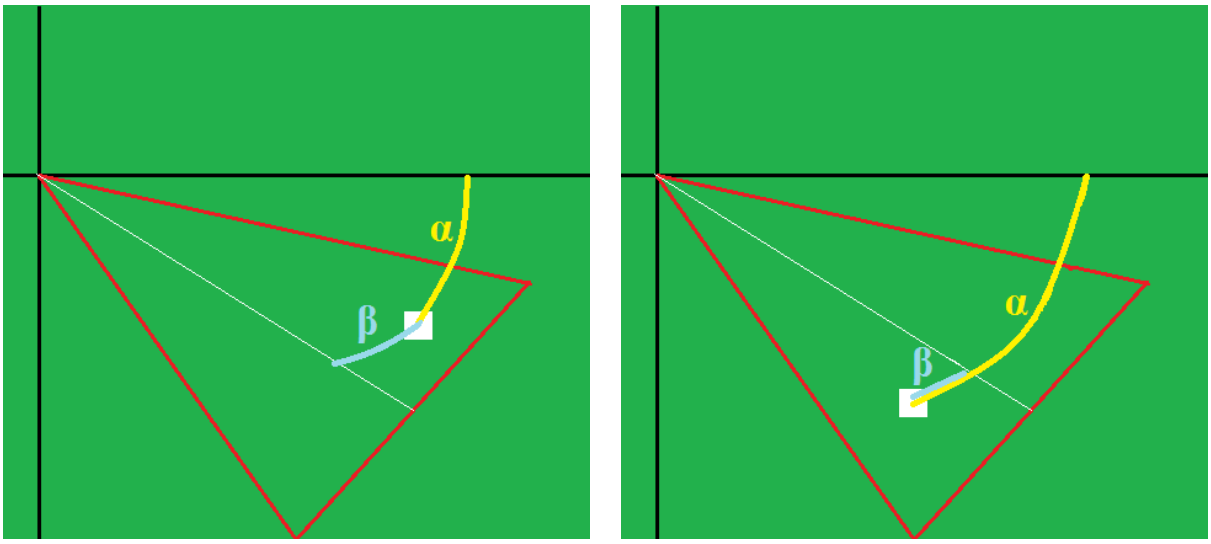
3. Seja β o ângulo entre a *feature* e a bissetriz do ângulo da visão do robô, α o ângulo entre a *feature* e a linha horizontal que define o zero da orientação do mundo.

4. Temos 2 possíveis casos, o primeiro, quando a *feature* está no lado esquerdo da imagem e o segundo, quando está a direita. Para o primeiro, como ilustra a Figura 4.5a.

$$\theta_1 = \alpha + \beta \tag{4.2}$$

e o segundo, mostrado na Figura 4.5b.

$$\theta_2 = \alpha - \beta \tag{4.3}$$



(a) Feature a esquerda da bissetriz (b) Feature a direita da bissetriz

Figure 4.5: Estimação da orientação do robô em relação ao mundo com base na observação de uma feature pela câmera

Um problema da abordagem utilizada, é que para determinado conjunto de elementos avistados, pode-se encontrar várias posições prováveis, nestes casos, para que o melhor candidato seja selecionado, usou-se como heurística o fato de que o robô é lento e em geral ele deve estar em algum lugar próximo de sua leitura anterior. Portanto, para essa abordagem, a posição escolhida como representante na etapa de correção é o ponto mais próximo de sua posição no instante anterior.

Assim, a função h apresenta a seguinte forma:

$$h(x_k, u_k) = \begin{bmatrix} x_k^g - dx \\ y_k^g - dy \\ tan^{-1}\left(\frac{y_k^g - dy}{x_k^g - dx}\right) \end{bmatrix}, \tag{4.4}$$

onde x_k^g, y_k^g é a posição global do elemento encontrado pela câmera nos eixos x e y respectivamente, dx é a distância informada da câmera entre a *feature* e o robô no eixo x e dy é a distância informada da câmera entre a *feature* e o robô no eixo y.

Com isso, tem-se uma pose (x, y, θ) na etapa de predição e outra na etapa da correção, onde a partir das equações de $f(\hat{x}_{k-1}, u_k)$ e $h(x_k, u_k)$ cria-se as matrizes F e H, que foram definidas por (3.37) e (3.41) respectivamente.

$$F = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.5}$$

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.6}$$

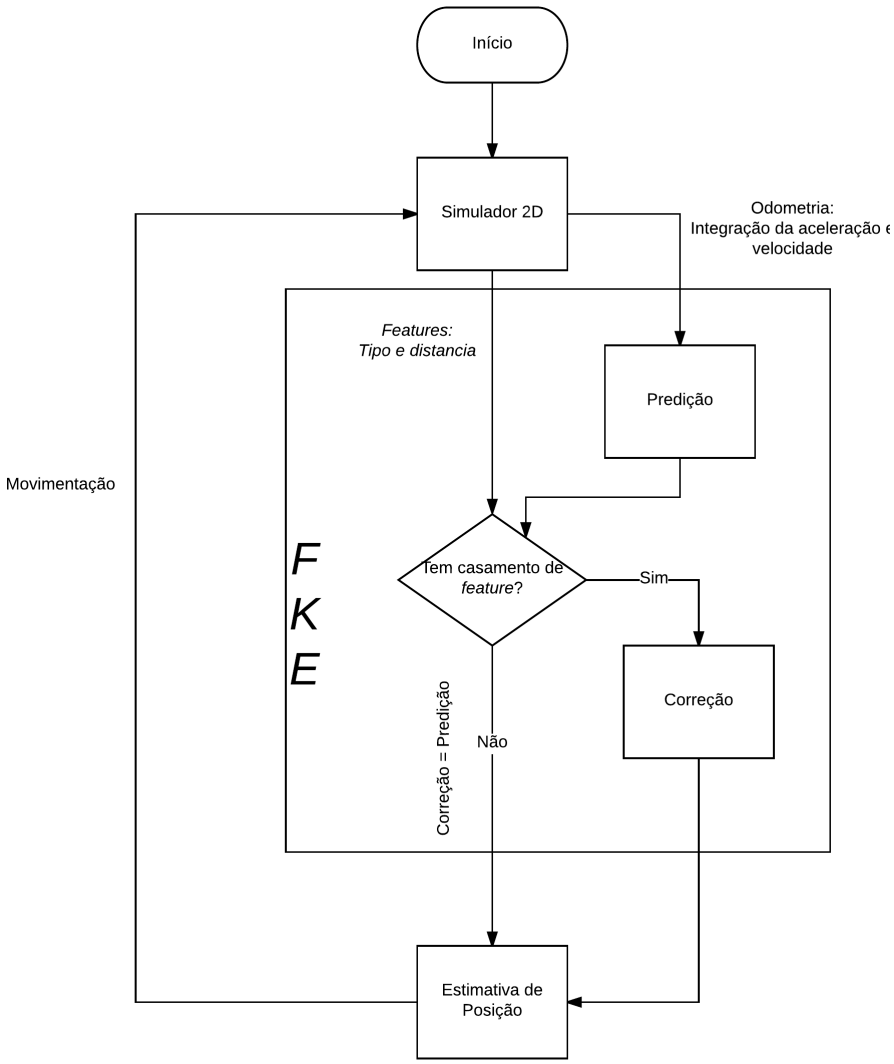


Figura 4.6: Fluxograma do processo de localização baseado no FKE

Logo uma vez terminado o equacionamento das matrizes F e H já se calculou todas as equações necessárias para a implementação da fusão sensorial através do FKE.

Uma vez apresentado o funcionamento método de localização é necessário destacar que nesse processo existem alguns momentos em que a correção não pode ser feita, no caso de quando o robô entra na zona morta do campo. A zona morta, pode ser definida como posições do campo em que o robô é incapaz de observar qualquer elemento do qual ele possa retirar alguma informação de sua provável posição. Nestes casos, a correção nada mais se torna do que a própria predição, o que aumentará o erro para as próximas leituras.

Para fechar a descrição dos métodos implementados nesse trabalho, resumiu-se em forma de fluxograma, conforme mostrado a Figura 4.6, as etapas-chave do processo de localização, onde todo o resto do processo que foi ocultado deste diagrama pode variar conforme a implementação.

5 RESULTADOS

Uma vez apresentada toda a fundamentação teórica que foi utilizada no trabalho e a motivação que levou ao mesmo, neste capítulo será exposto os resultados obtidos. Começando como a navegação inercial do robô se comporta, informações essas que foram obtidas em ensaios reais, depois irá-se apresentar o funcionamento do simulador 2D e por fim, os resultados obtidos pela implementação da estratégia de localização no simulador, bem como as análises a fim de se estudar o impacto das detecções de cada tipo de *feature.*

5.1 NAVEGAÇÃO INERCIAL

A navegação inercial, como já dito na Subseção 3.6 é a parte responsável por coletar os dados da unidade inercial e tratá-los de maneira a se obter informação a respeito da posição e velocidade do robô. Então inicialmente foi criado um módulo em C++ para que se pudesse observar o comportamento da navegação inercial no robô real e assim analisar os problemas que a integração do erro causa de fato na navegação inercial. Um módulo na arquitetura do NAO, nada mais é que um tarefa preemptiva chamada de tempos em tempos, é necessário fazer isso, uma vez que o processador do robô possui apenas um núcleo, não sendo assim possível realizar processamentos em paralelo. Durante uma partida de futebol, para o bom desempenho do jogador, a equipe UnBeatables desenvolveu vários módulos, que devem rodar em paralelo, são eles:

- Locomoção: Responsável por toda a movimentação.
- Visão: Módulo responsável por capturar os dados da câmera e fazer todo o processamento de imagens, além de estimar a distância das *features* até ao robô.
- Comportamento: Módulo que define a estratégia adotada nas partidas, ou seja, como o robô irá se comportar baseado nas informações que ele obteve do mundo através dos outros módulos.
- Comunicação: Módulo que envia e recebe as informações via wifi para o *Game Controller*.

Uma vez que foi possível ler as informações da IMU, foi implementado no módulo navegação inercial a evolução dinâmica do sistema conforme descrito em (3.43) - (3.49). Os cálculos envolvendo quatérnios foram implementados utilizando a biblioteca DQrobótics [14]. Assim foram feitos vários testes onde é interessante destacar alguns. Primeiramente utilizando funções da NAOqi que são disponibilizadas pelo fabricante, fez-se o robô executar uma movimentação em "L", ou seja, ele deveria se deslocar para frente por um metro, girar 90º no sentido anti-horário e se deslocar por mais im metro, enquanto era feita a navegação inercial do percurso, onde o resultado pode ser conferido na Figura 5.1.

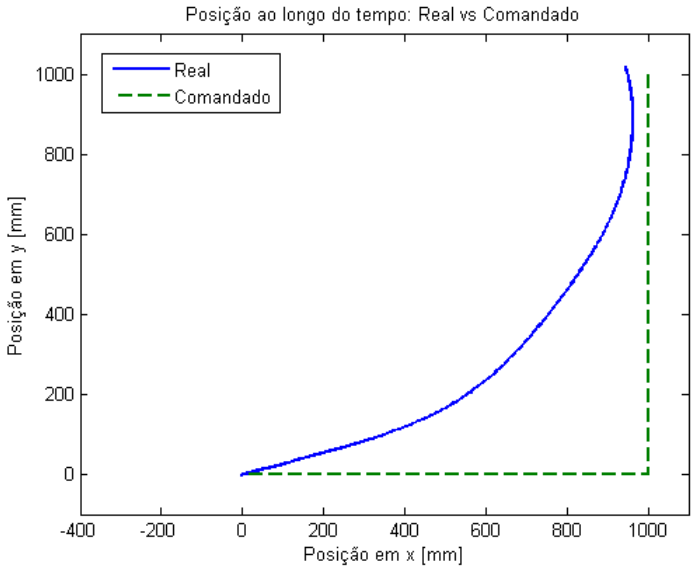


Figura 5.1: Deslocamento real do robô *vs*. deslocamento esperado

Na Figura 5.1 é possível ver uma discrepância entre o trajeto feito e o indicado pela navegação inercial. Essa discrepância pode ser atribuída principalmente ao processo de integração das leituras e o próprio escorregamento dos pés, pois o sensor possui um erro associado a sua medida e ao longo do tempo este erro vai se acumulando, aumentado a imprecisão dos resultados cada vez mais, enquanto o robô se locomove para frente, bem como executa giros, é notável que ele não percorre exatamente o trajeto comandado, motivo esse que torna a navegação inercial não confiável na maioria dos casos em malha aberta.

Este erro pode ser conferido em trajetos maiores e para mostrar isso, foi feito um teste com o NAO executando um percurso em forma quadrada. Neste segundo percurso, o robô deve se deslocar por um metro para frente, girar 90º no sentido anti-horário, repetindo esse processo mais 3x, formando assim um trajeto quadrado ao longo do tempo. Trajeto pode ser conferido na figura 5.2. Comparando as figuras, também observa-se que o erro não é sistemático e a odometria além de ter pouca acurácia também é imprecisa.

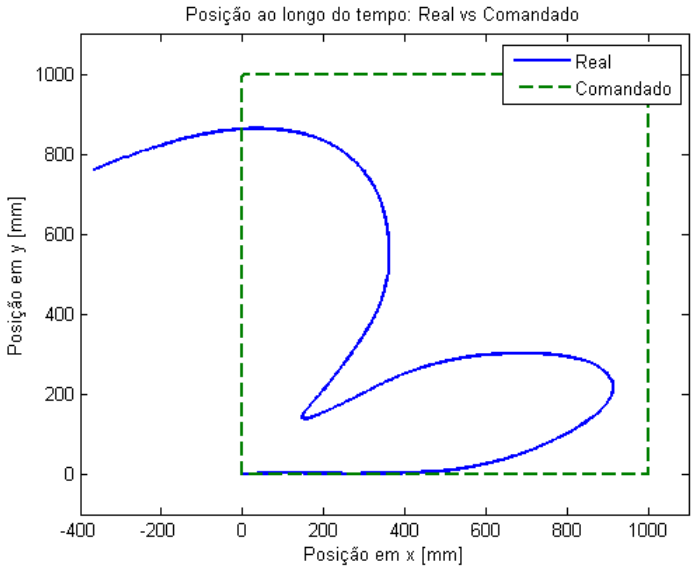


Figura 5.2: Deslocamento estimado pela navegação inercial, trajeto comandado e trajeto observado

Onde durante a execução deste teste foi notado novamente os escorregamentos ao longo do percurso que pode alterar sua trajetória em malha aberta, além de erros associados a movimentação das juntas que atrapalham da execução da movimentação, logo o trajeto em forma de caixa, acabou saindo de outra maneira. Em seu segundo giro, o escorregamento somado a integração de erro, ocasionou um giro maior que o comandado, alterando assim a forma desejada, o trajeto efetivo aproximou-se de uma forma triangular como ilustrado na figura 5.3.

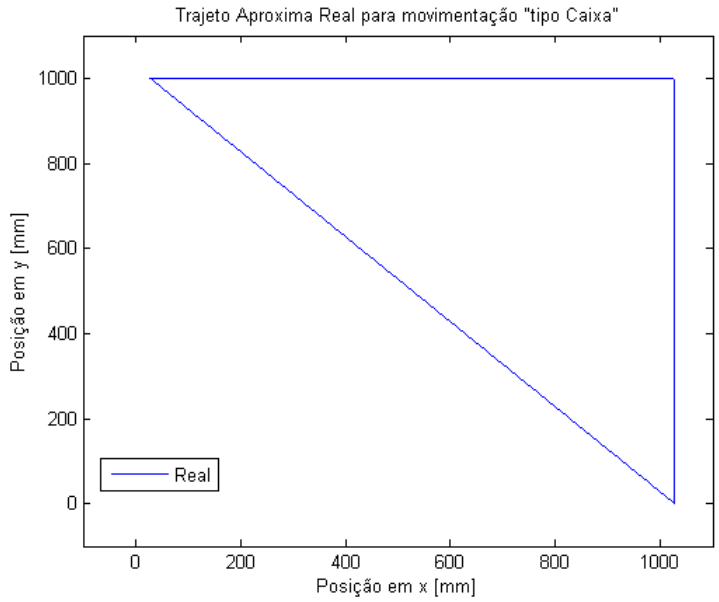


Figura 5.3: Deslocamento errôneo do robô inspecionado visualmente

Observando o trajeto proposto e o trajeto efetivo, pode-se chegar as seguintes conclusões:

1. A navegação inercial nos primeiros segundos, assim como ensaio anterior, é relativamente con-

fiável, apresentando uma boa leitura.

2. Com o passar do tempo, o acumulo de erros, como esperado, é tão alto, que gera resultados absurdos, evidenciando assim a importância de fazer a correção destes valores, função essa que será desempenhada pelo filtro de Kalman estendido, onde uma vez que tenha-se a informação corrigida em períodos relativamente curtos, é possível assim diminuir os erros do processo e ter uma boa estimativa da posição do robô .

5.2 PROJEÇÃO

Para validar a estimação de distância implementada no módulo Visão Computacional, foi testado o método implementado que é baseado em [16] para realizar a projeção do sistema de coordenadas da imagem para o mundo real. Este método parte do mesmo princípio do modelo pinhole de câmera, usando apenas semelhança de triângulo que funciona da seguinte forma:

1. Inicialmente deve-se posicionar um objeto qualquer de tamanho conhecido W, neste caso de teste, foi posicionada uma bola, por apresentar boa manuseabilidade, a uma distância D do robô.

2. O próximo passo é obter o tamanho do objeto P em pixel.

3. Uma vez conhecido o tamanho real W, o tamanho na imagem P e a distância do objeto ao robô D, é possível calcular uma distância focal aparamente, que só é válida para este objeto:

$$F = (P * D)/W \tag{5.1}$$

4. E assim, tendo a distância focal aparente deste objeto, pode-se calcular sua distância vertical, ou seja, relativa ao eixo X do robô:

$$D' = F * W/P \tag{5.2}$$

5. Para calcular a distância relativa ao robô no eixo Y, ou seja, quanto para direita ou para a esquerda do robô estava a bola, foi feito da seguinte forma:

$$y = (posição_x * (u - ppx)/distancia\_focal\_aparente)/G \tag{5.3}$$

onde $posição_x$ é o valor calculado no passo 4, *u* é o centro da posição horizontal em pixel do objeto na imagem, *ppx* é um *offset* que desloca a origem do centro e coordenadas da imagem para o canto esquerdo superior da tela, gerando o sistema de coordenadas u e v da imagem, sendo *ppx* o *offset* relativo ao eixo x da imagem. Ao realizar os testes, notou-se uma certa imprecisão nos valores obtidos. Esta imprecisão é influenciada pelo fato da câmera que está no robô fique a uma certa altura do chão, uma vez que ela se encontra na cabeça do NAO, por isso foi introduzido um ganho *g* para que pudesse ser feito esta compensação.

Na Tabela 5.7 encontra-se os resultados obtidos para estimativa da distância da bola utilizando este algoritmo.

Tabela 5.1: Posição da bola

Posição Real da Bola (mm)		Posição Estimada da Bola (mm)	
x	y	x	y
2000	0	2150	0
1000	0	1000	0
500	300	490	285
300	50	298	55
100	1000	110	1057

Para as demais features seria necessário fazer uma calibração semelhante, para se obter as estimativas de distância.

5.3 SIMULADOR 2D

Dado alguns problemas encontrados no módulo de Visão Computacional, a implementação do simulador foi uma boa maneira de se contornar esses problemas, porém é claro que apresenta seus prós e contras, como é descrito a seguir:

- Vantagens:
  - Maior facilidade para se testar o algoritmo proposto, uma vez que é fácil definir a posição inicial do robô no mundo, bem como é muito mais prático saber com exatidão a posição "real" do mesmo, para que se possa comparar a posição "real" com a posição estimada, sem que seja necessário fazer marcações ou medições no campo físico.
  - Por ser um ambiente simulado, é possivel contornar os problemas de visão computacional, já que esse não é o foco deste trabalho e assim simplesmente utilizar os resultados esperados destes procedimentos.
- Desvantagens:
  - Por ser um ambiente controlado, sabe-se que o mesmo pode não retratar a realidade em alguns casos.
  - A simulação é fortemente baseada na cinemática do robô excluindo a influência da dinâmica

No Capítulo anterior, foram apresentadas quais funcionalidades o simulador seria capaz de executar e agora será mostrado como usar estas funções no simulador bem como mostrar os resultados das mesmas.

5.3.1 Posicionamento do NAO

No Matlab, para que posicione o NAO no mundo, basta criar o objeto *robot* que recebe como argumento a pose (x, y, θ), onde deseja-se posiciona-lo, sendo x e y em mm e θ em graus. Por exemplo, para colocar o robô na origem, basta utilizar o comando: *nao = robot(0,0,0),* Onde *nao* é o objeto *robot* . Com a criação desse objeto, a saída pode ser conferida na Figura 5.4.

```
Robot with properties:

            duracao: 100
        amostragem: 0.5000
        pos_x_real: 0
        pos_y_real: 0
        theta_real: 0
         pos_est_x: []
         pos_est_y: []
     pos_est_theta: 0
         pos_x_odo: 0
         pos_y_odo: 0
         theta_odo: 0
     pos_x_odo_ant: []
     pos_y_odo_ant: []
     theta_odo_ant: []
         vel_x_odo: 0
         vel_y_odo: 0
       vel_ang_odo: 0
                xq: [1x5232 double]
                yq: [1x5232 double]
            quinas: [8x2 double]
 quinas_encontradas: []
       dist_quinas: []
        rot_quinas: []
            traves: []
       dist_traves: []
        rot_traves: []
            linhas: []
       dist_linhas: []
        rot_linhas: []
       achoucentro: 0
       dist_centro: []
               rot: []
```

Figura 5.4: Criação do objeto *robot* no simulador

O objeto *nao* criado, possui todas as propriedades que serão necessárias para a estimação da posição, como mostra a Tabela 5.2.

Tabela 5.2: Propriedades do objeto robot

Atributo	Descrição
duracao	Indica a duração, em segundos, da última coleta de dados.
amostragem	Taxa de amostragem dos dados.
`pos_x_real`	Posição verdadeira no eixo x em relação ao mundo.
pos_y_real	Posição verdadeira no eixo y em relação ao mundo.
theta_real	Orientação verdadeira em relação ao mundo.
pos_est_x	Posição no eixo x estimada pela visão.
pos_est_y	Posição no eixo y estimada pela visão.
pos_est_theta	Orientação estimada pela visão.
pos_x_odo	Posição no eixo x obtido pela navegação inercial.
pos_y_odo	Posição no eixo y obtido pela navegação inercial.
theta_odo	Orientação obtido pela navegação inercial.
pos_x_odo_ant	Valor da navegação inercial no instante anterior relativa ao eixo x.
pos_y_odo_ant	Valor da navegação inercial no instante anterior relativa ao eixo y.
theta_odo_ant	Valor da navegação inercial no instante anterior relativa a orientação.
vel_x_odo	Velocidade no eixo x obtida pela navegação inercial.
vel_y_odo	Velocidade no eixo y obtida pela navegação inercial.
vel_ang_odo	Velocidade angular obtida pela navegação inercial.
xq	Vetor de pontos que representa o mapa.
yq	Vetor de pontos que representa o mapa.
quinas	Vetor que contém as coordenadas das interseções de linhas no campo.
quinas_encontradas	Coordenada da(s) quina(s) econtrada(s).
dist_quinas	Vetor que contém a distância do robô até a(s) quina(s) encontrada(s).
rot_quinas	Vetor que contém o(s) ângulo(s) entre o robô e a(s) quina(s) encontrada(s).
traves	Vetor que contém a(s) coordenada(s) da(s) trave(s) do gol no campo.
dist_traves	Vetor que contém a distância do robô até a(s) trave(s) encontrada(s).
rot_traves	Vetor que contém o(s) ângulo(s) entre o robô e a(s) trave(s) encontrada(s).
linhas	Vetor que contém a coordenada inicial e final de cada linha encontrada.
dist_linhas	Vetor que contém a(s) distância(s) do robô até a(s) linha(s) encontrada(s).
rot_linhas	Vetor que contém o(s) ângulo(s) entre o robô e a(s) linha(s) encontrada(s).
achoucentro	Boleano que indica se o centro do campo está sendo avistado.
dist_centro	Distância do robô até o centro de campo.
rot	Rotação entre robô e o centro de campo.

5.3.2 Movimentação e angulo de visão do NAO

Existem quatro funções de movimentação implementadas no simulador, que são chamadas aleatoriamente onde ao final de cada movimentação é gerada uma imagem do mundo contendo a posição real do robô e o campo de visão, onde o mundo é representado pelas linhas na cor azul e os elementos dentro do campo de visão ficam na cor vermelha. A Figura 5.5 ilustra uma determinada posição do NAO, as Figuras 5.6a e 5.6b mostra a execução da função de avanço e de recuo, respectivamente e por fim, a Figura 5.7 mostra as funções de rotação positiva e rotação negativa, a partir da posição inicial da Figura 5.5.

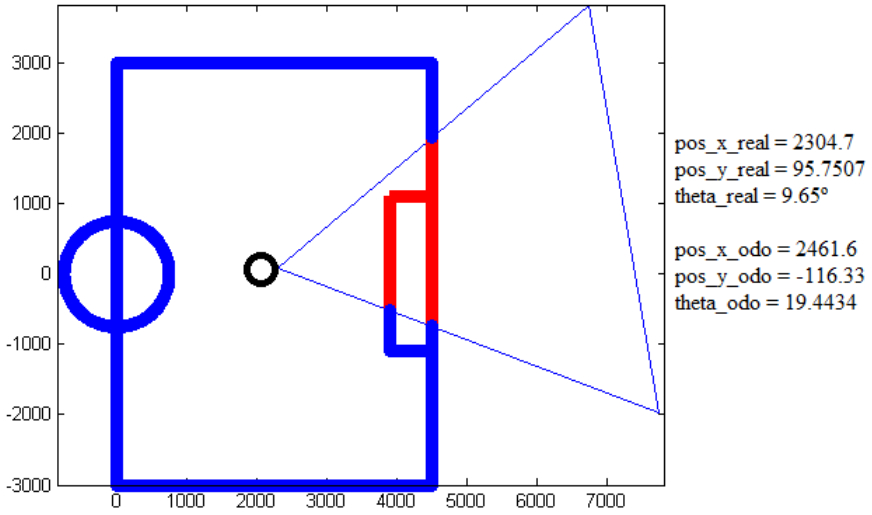
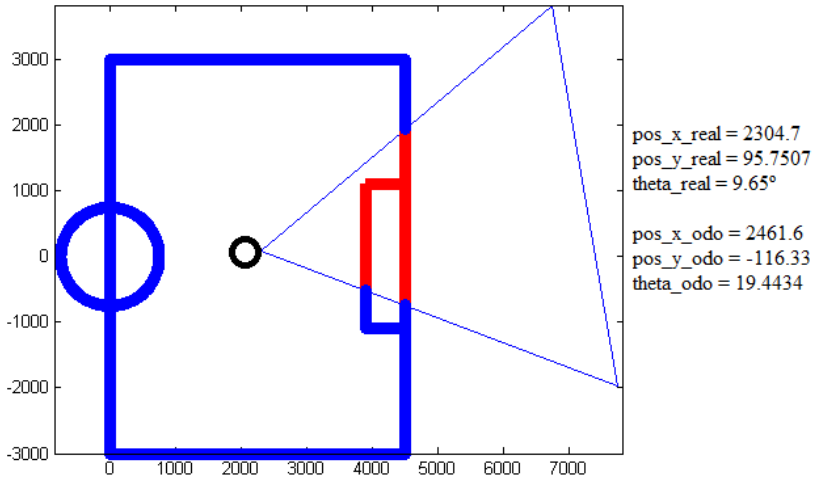
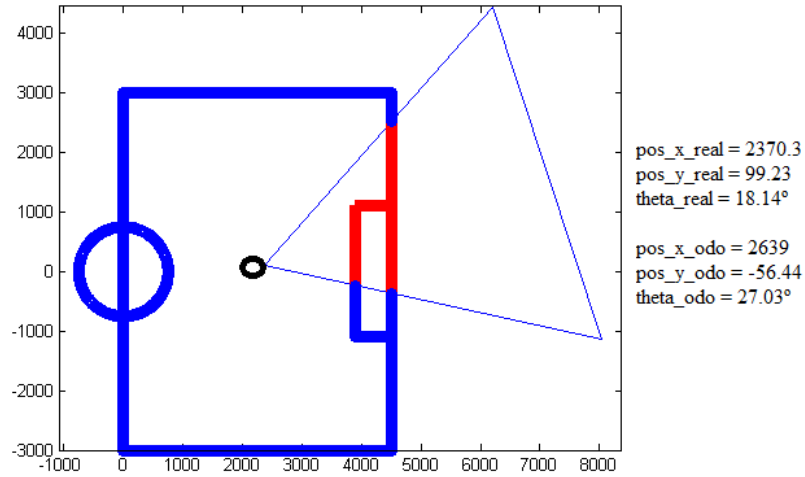


Figura 5.5: Posição base para ilustração das funções de movimento implementadas no simulador

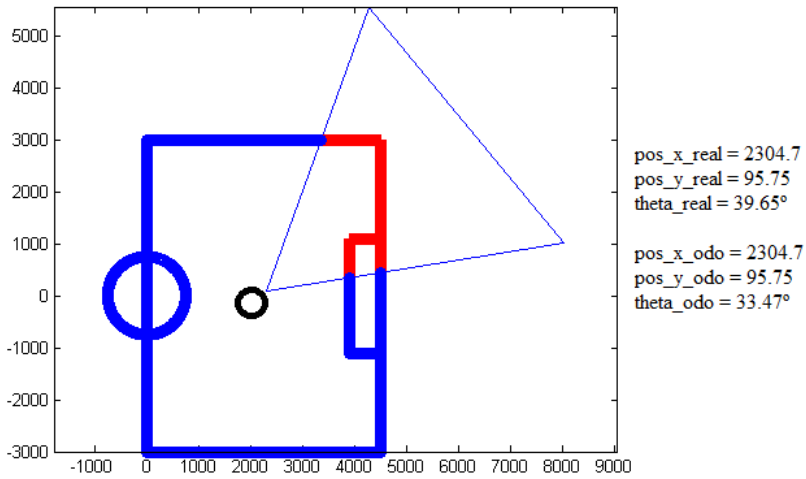


(a) Movimento Avanço

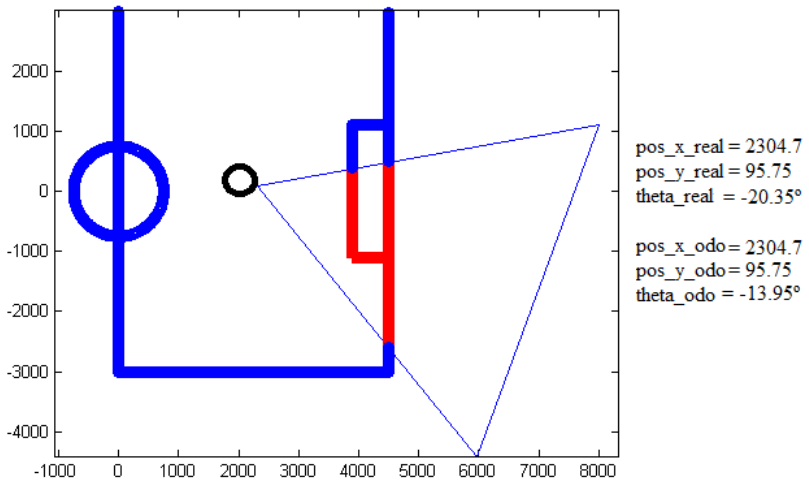


(b) Movimento Recuo

Figura 5.6: Funções de movimentação avanço e recuo a partir da posição base



(a) Movimento de rotação positiva



(b) Movimento de rotação negativa

Figura 5.7: Funções de rotação a partir da posição base

5.4 FILTRO DE KALMAN ESTENDIDO

No capítulo 3 já foi abordado as fundamentações teóricas necessárias para a modelagem do filtro. O vetor de estados relativos a predição:

$$x_k = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} x_{k-1} + T.v_x[k] \\ yk - 1 + T.v_y[k] \\ \theta_{k-1} + T.\omega[k] \end{bmatrix} + w(k), \tag{5.4}$$

onde x_{k-1}, y_{k-1}, θ_{k-1}, são os valores da posição do robô no instante anterior, T é o período de amostragem e v_x, v_y, ω são as velocidades nesse instante do robô e $w(k)$ é uma gaussiana normal de média 0 e variância Q.

Já a correção, pode ser definida como:

$$y_k = \begin{bmatrix} x[k] \\ y[k] \\ \theta[k] \end{bmatrix} = \begin{bmatrix} x_k^g - dx \\ y_k^g - dy \\ tan^{-1}\left(\frac{y_k^g - dy}{x_k^g - dx}\right) \end{bmatrix} + v[k], \tag{5.5}$$

onde $x_k^g,\ y_k^g$ é a posição global do elemento encontrado pela câmera nos eixos x e y respectivamente, dx é a distância informada da câmera entre a feature e o robô no eixo x e dy é a distância informada da câmera entre a feature e o robô no eixo y e v[k] é uma gaussiana normal de média 0 e variância R.

A partir destas equações podemos construir as matrizes F e H:

$$F = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{5.6}$$

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{5.7}$$

5.4.1 Resultados da implementação do filtro de Kalman estendido no simulador

Primeiramente, para se observar o desempenho do filtro, colheu-se por 100 segundos a movimentação do NAO em meio campo de futebol, com um período de amostragem T = 0.5s. Então utilizando os dados de navegação do robô obtidos pela simulação, foi feita uma análise *offline*, aplicando a técnica do filtro de Kalman estendido usando como valores das matrizes Q e R:

$$Q = \begin{bmatrix} 0.5^2 & 0 & 0 \\ 0 & 0.5^2 & 0 \\ 0 & 0 & 0.7^2 \end{bmatrix}, \tag{5.8}$$

$$Q = \begin{bmatrix} 0.15^2 & 0 & 0 \\ 0 & 0.15^2 & 0 \\ 0 & 0 & 0.15^2 \end{bmatrix}, \tag{5.9}$$

e seu resultado pode ser conferido na Figura 5.8.

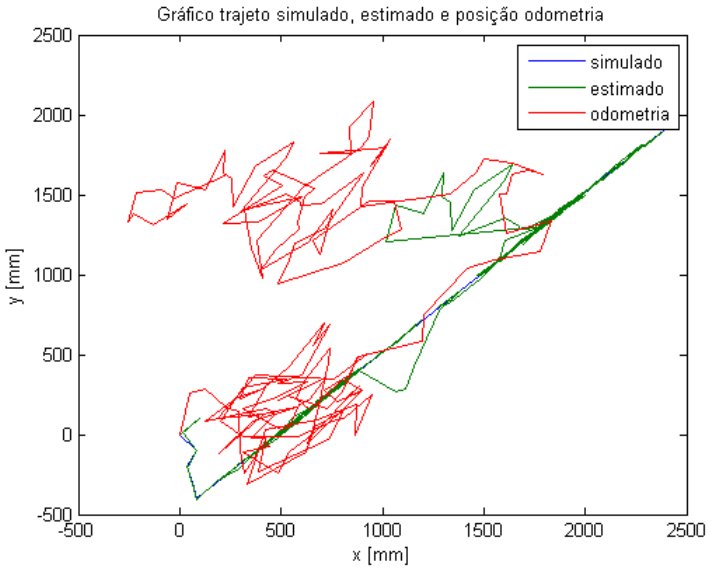


Figura 5.8: Trajetos simulado, estimado pelo FKE e estimado pela navegação inercial em uma execução

Onde o trajeto em azul seria o caminho percorrido em simulação pelo NAO, já a linha vermelha representa os dados de navegação colhidos pela navegação inercial, e por fim, o trajeto na cor verde, mostra o trajeto estimado pelo filtro. Vale ressaltar que nessa simulação em especial, dos 200 pontos amostrados, 77 pontos não obtiveram correção da câmera, ou seja, em 77 pontos (38.5%), não foi possível identificar elementos marcantes na imagem que auxiliassem na correção e mesmo assim foi possível estimar muito bem a posição em X e Y do robô no mundo, como pode ser melhor visualizado nas Figuras 5.9 e 5.10.

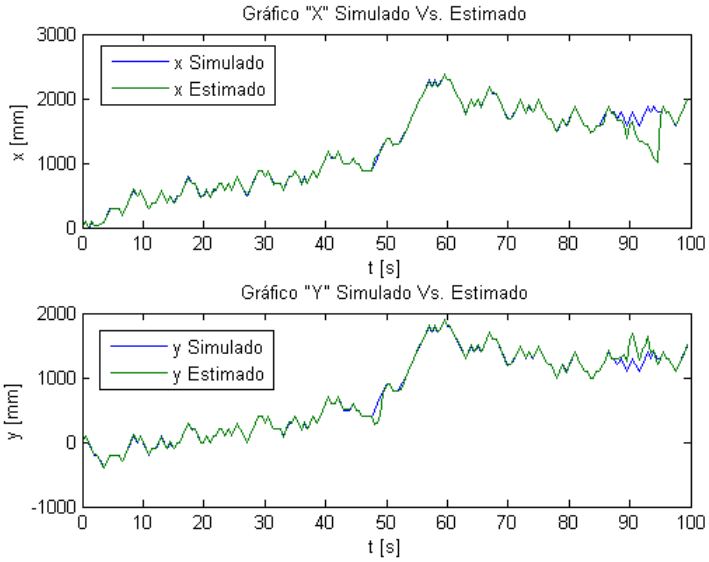


Figura 5.9: Posição x e y simulada e posição estimada ao longo do tempo

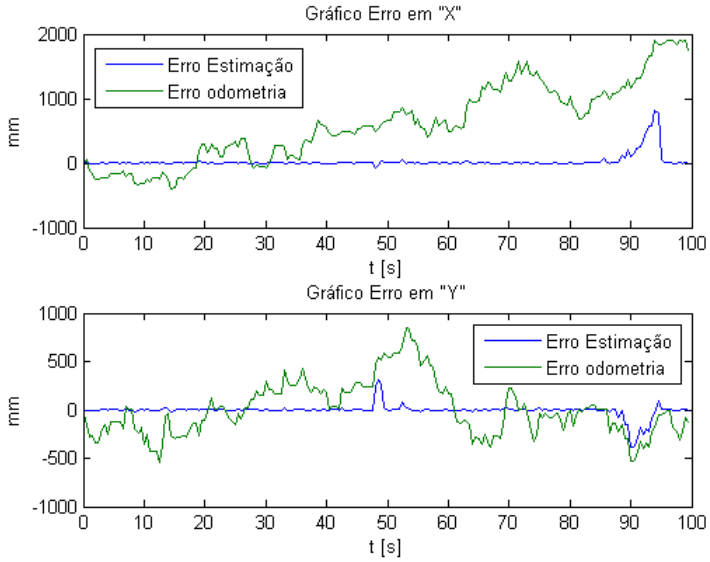


Figura 5.10: Comparação dos erros nos eixos x e y da estimação e da navegação inercial ao longo do tempo em simulação

Além de conseguirmos estimar a posição em x e y, a orientação é muito importante, já que sabendo a orientação correta, o robô pode direcionar-se corretamente para o gol adversário, podendo assim poder pontuar na competição. Ao realizarmos os testes, notamos que a orientação é muito mais sensível, em outras palavras, nos momentos em que não ocorrem a correção pela informação da câmera e consequentemente tendo nossa estimação igual a predição, ocasiona um erro muito mais acentuado do que o observado na posição x,y. No ensaio mostrado na Figuras 5.8, tivemos a predição executada corretamente em 61.5% do tempo, caso adotemos o método de avaliação usado por Whelan T., Studli S., McDonald J., Middleton R. H., em [9] que considera como erro máximo aceitável, uma estimativa que esteja em até 30cm da posição correta, e uma orientação correta como sendo aquela que possui um erro inferior a 15º, usando então essa métrica, obtivemos uma precisão de 95% no posicionamento x,y e já no caso da orientação 81.5% das vezes.

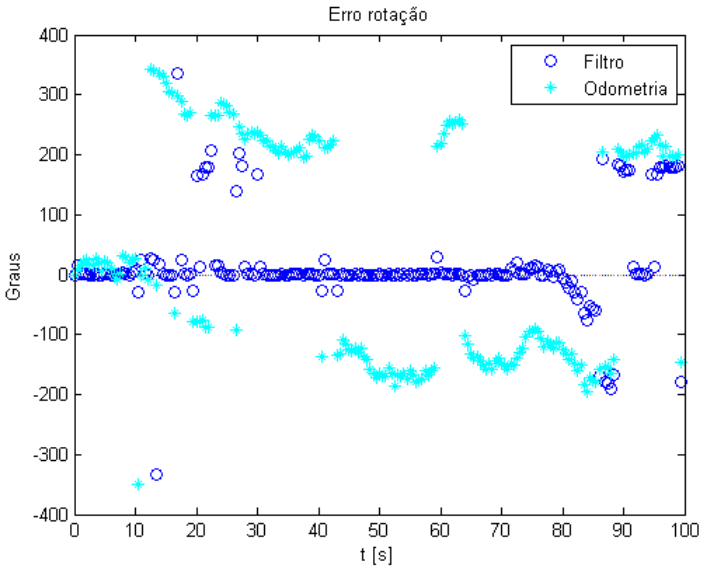


Figura 5.11: Erro da orientação do NAO ao longo do tempo em simulação

5.4.2 Impacto dos diferentes tipos de *features* no processo da localização

Para analisarmos a relevância das *features* no processo de localização, fizemos uma análise usando três trajetórias diferentes, onde em cada uma delas variamos na etapa de correção quais *features* seriam levadas em conta nessa parte do processo, tendo então quatro casos para cada trajeto:

1. Detecção de todas as *features*;
2. Apenas do traves do gol;
3. Apenas o centro de campo;
4. Apenas interseção de linhas;

onde para cada trajeto proposto, foram executadas dez vezes a mesma trajetória. Para este ensaio foram escolhidas três posições inciais e apresentaram o seguinte percurso:

1. Posição no meio de campo (0,0,0): Neste ensaio o robô partia da origem e se deslocava na maior parte do tempo em direção ao gol;
2. Posição do Goleiro (4000,0,180): Neste ensaio o robo partia da posição de goleiro e se deslocou na diagonal, executando alguns giros esporadicamente;
3. Posição inicial pobre de informação (2000, -1500, 90): Partindo de uma posição onde não é possível detectar muitos elementos do campo o robo seguiu uma trajetória reta, executando muitos giros nos 100s de simulação;

Os trajetos 1, 2 e 3 possuem sua posição inicial marcada, bem como a direção predominante do trajeto executado ilustrado na Figura 5.12 e os resultados médios da estimação de cada trajeto é mostrado nas

Tabela 5.3: Análise da eficiencia do processo de estimação baseado na detecção de todas as *feature*s em conjunto

Posição Inicial	% Acurácia média		% Correção média	RMS do erro médio	
	xy	θ		xy	θ
Trajeto 1	97,3	71,5	87,2	113,3334	113,6941
Trajeto 2	98	70,5	86,5	111,2289	139,733
Trajeto 3	95,5	79	74,5	356,41	164,1

Tabela 5.4: Análise da eficiencia do processo de estimação baseado na detecção das traves do gol

Posição Inicial	% Acurácia média		% Correção média	RMS do erro médio	
	xy	θ		xy	θ
Trajeto 1	64,3	69,7	40,7	915,53	158,60
Trajeto 2	58,5	56,5	27,5	566,37	200,05
Trajeto 3	41,5	87	23,5	1177,26	260,13

Tabelas 5.3 - 5.6 que contém a acurácia da estimação de posicionamento, orientação e a porcentagem de vezes que a etapa de correção do FKE foi executada corretamente, bem como o valor médio RMS referente ao erro.

No primeiro caso, como esperado, o filtro com a análise completa apresentou um bom resultado, neste mesmo ensaio, a eficiência da estimação contando apenas com as detecções do gol e do meio de campo mostram baixo desempenho, como era de se esperar já que a partir da posição inicial, leva-se um tempo até avistar-se as traves do gol e nesse trajeto, os únicos momentos em que o meio de campo é possível ser visto, dependendo da posição, é quando a orientação está no segundo ou terceiro quadrante. Para esses trajetos, uma surpresa foi os resultados obtidos no último caso, que apresentaram um precisão média maior que 90% de estimativa.

No segundo e terceiro caso, a estimação com o processo completo, manteve sua qualidade, porém, ocorreu inversamente o contrário do primeiro caso em relação a estimação da posição baseada somente na intersecção de linhas como método de correção, onde justamente de se ter pouca informação contidas nestes ensaio a estimação teve como base a navegação inercial e como isoladamente as quinas pouco são representativas, mesmo quando elas apareciam, não eram o suficiente para corrigir todo o acúmulo de erro nos instantes passados, onde também vale lembrar que, quando se tem mais que uma posição provável, aquela escolhida é a mais próxima do instante anterior, sendo assim nesse ponto em que o erro ataca na etapa da correção nesses casos.

Tabela 5.5: Análise da eficiencia do processo de estimação baseado na detecção do meio de campo

Posição Inicial	% Acurácia média		% Correção média	RMS do erro médio	
	xy	θ		xy	θ
Trajeto 1	36,1	29,9	20,5	950,02	229,85
Trajeto 2	42,5	31	23	1141,56	201,23
Trajeto 3	38	47,5	15	786,19	259,27

Tabela 5.6: Análise da eficiencia do processo de estimação baseado na detecção das quinas

Posição Inicial	% Acurácia média		% Correção média	RMS do erro médio	
	xy	θ		xy	θ
Trajeto 1	93	72,5	63	199,39	155,18
Trajeto 2	0	48,9	69,6	6139,12	194,42
Trajeto 3	0	54,6	69,4	6026,16	186,49

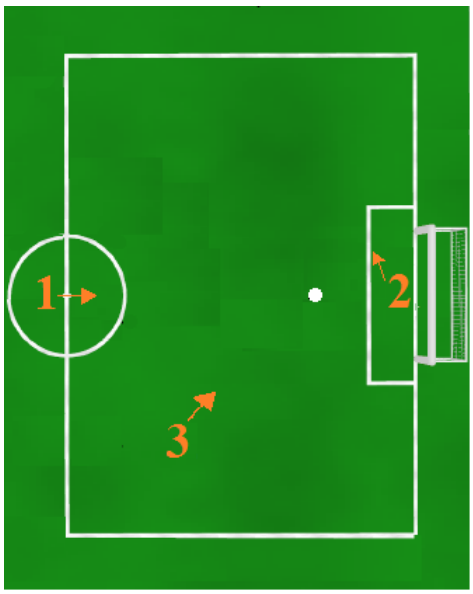


Figura 5.12: Posição inicial e direções aproximadas do trajeto nos testes

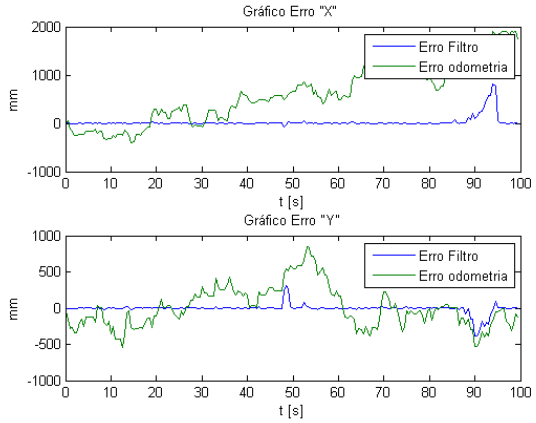
5.5 LOCALIZAÇÃO EM CAMPO COMPLETO

Todo o desenvolvimento do trabalho até então, focou-se até então apenas na localização considerando metade do campo de futebol, onde obteve-se resultados positivos dentro do escopo proposto. Porém como análise final, foi aplicado o algoritmo desenvolvido no campo completo. Para a aplicação no campo completo, foram necessárias apenas as seguintes alterações:

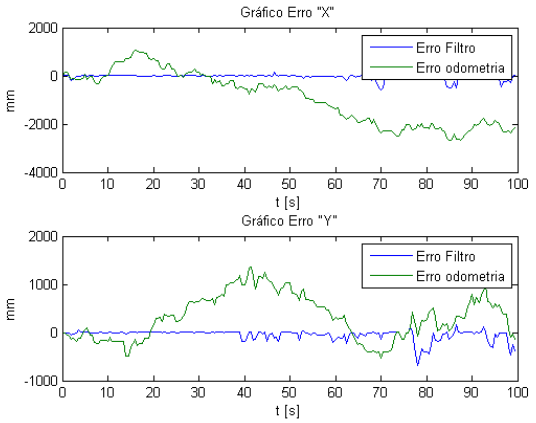
- Primeiramente, foi criado o campo completo no simulador adicionando os pontos pertencentes a outra metade no mapa já existente;
- Inseriu-se as informações pertinentes as coordenadas das novas interseções de linhas e do novo gol;
- E por fim, foi redefinida a função que monitora se o robô saiu do campo ou não, pois antes, as posições possíveis do NAO em campo era de 0 a 4500 no eixo x e de 3000 a -3000 no eixo y. No campo completo alteramos os limites no eixo x para -4500 a 4500;

Para que se possa comparar de maneira mais clara o resultado da aplicação no campo completo, executou-se o mesmo trajeto nos dois ambientes dez vezes, o primeiro teste será em meio campo e o

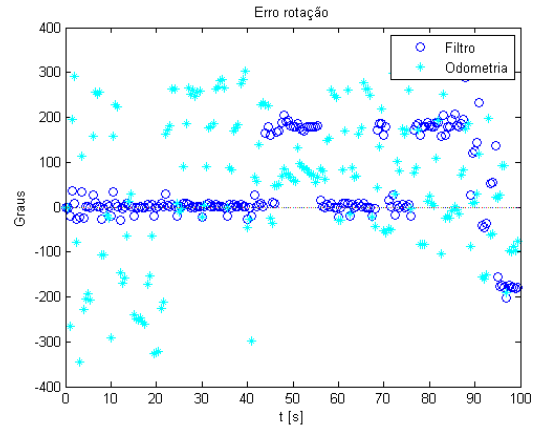
segundo no campo completo. A Figura 5.13 mostra os erros no posicionamento XY e de θ ao longo do tempo.



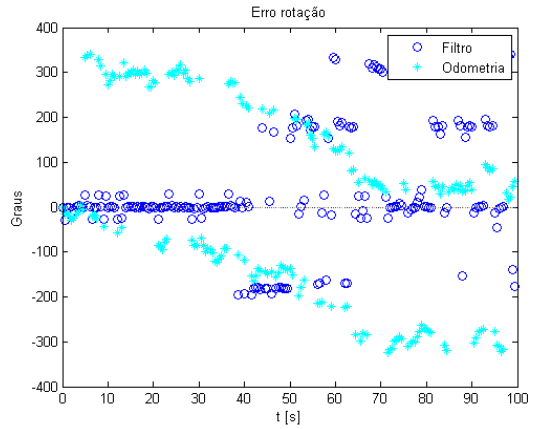
(a) Erro xy em meio campo



(b) Erro xy no campo completo



(c) Erro da rotação em meio campo



(d) Erro da rotação no campo completo

Figura 5.13: Comparação da estimação em um mesmo trajeto: meio campo x campo completo

E já usando as métricas de avaliação de resultados definidas anteriormente, foi elaborada a Tabela 5.7 com a precisão média de cada situação.

Tabela 5.7: Comparação da eficiencia da localização entre meio campo e campo completo

Posição Inicial	Meio Campo					Campo Completo				
	%Precisão		%Correção	RMS médio		%Precisão		%Correção	RMS médio	
	xy	θ		xy	θ	xy	θ		xy	θ
Origem do campo	95	74	88	129,64	105,42	88	67	85	332,58	164,1

Neste caso mostrado, pode-se observar que para meio campo, o algoritmo apresentou um desempenho pior mas ainda satisfatório aos mostrados na Tabela **??**, apresentando resultados consistentes. Já para o caso de campo completo, a precisão do posicionamento xy diminuiu, algo justificável, já que existem mais possibilidades de posicionamento dada certa distância. E quanto a orientação, ambos os casos apresentaram resultados semelhantes.

6 CONCLUSÕES

Implementar a localização no ambiente da Standard Plataform League como pudemos ver, é realmente uma tarefa complexa, pois toda a ambiguidade presente nas informações apuradas do mundo real, associadas com a baixa capacidade de processamento do NAO é o que melhor representa as dificuldades do problema. Portanto, pensando em eliminar as ambiguidades, resolveu-se simplificar o problema, diminuindo o espaço de trabalho para apenas metade do campo de futebol, eliminando assim a possibilidade de o robô existir em dois lados do campo. Decisão essa que reduz, mas não eliminou completamente as ambiguidades, ainda presente quando por exemplo é avistado somente 1 trave do gol, ou um número reduzido de interseções de linha.

Com a baixa precisão da IMU disponível, juntamente com os problemas de movimentação que um robô humanoide apresenta, a navegação inercial mostrou-se pouco confiável, portanto inevitavelmente foi necessário implementar uma técnica de fusão sensorial, sendo a escolhida o filtro de Kalman estendido, ele não um filtro ótimo como o filtro de Kalman tradicional, mas é capaz de lidar com a não-linearidade do problema, coisa que a versão tradicional não é.

Após a implementação do filtro usando as estratégias apresentadas no trabalho, no média, obtivemos uma precisão no posicionamento x,y no campo superior a 90% e uma precisão média da orientação de 82% . Uma surpresa nos resultados, foi quando usada apenas as detecções das interseções de linhas, onde no começo do trabalho, pensava-se que essa informação sozinha era praticamente irrelevante, porém em alguns testes, ela mostrou-se surpreendentemente efetiva quando o robô se movimenta em locais ricos de informação, que foi o observado nos casos 1 da tabela 4.3. Já quando transitava em regiões mais pobres, como nos casos 2 e 3, ele foi completamente incapaz de estimar as posições, que era algo mais perto da ideia inicial.

Com isso, conclui-se que quanto mais o robô se movimenta perto da região do centro de campo e da área do gol, tendo eles como observação, melhor será a estimativa, sendo estas as áreas ricas de informação do trabalho. Em contrapartida, os piores casos observados são quando o jogador caminha orientado para as bordas superior e inferior do campo, regiões estas que só apresentam as linhas inteiras e algumas quinas. Isso reflete muito bem a qualidade da localização, pois uma vez as observações ao longo do tempo estejam boas, ao se ter alguns instantes apenas observando interseções de linhas, nossa estimativa ainda será boa, como visto nos casos 1. Porém o contrário também é valido, se para alguns trajetos, ou se o jogador ficar muitos instantes com uma observação pobre, a localização não será capaz de estimar sua posição até que se possa observar algo mais determinístico, como por exemplo as traves do gol.

Ao final do trabalho, foi aplicado o mesmo algoritmo no campo completo, ambiente típico da SPL, nele tivemos uma estimativa do posicionamento xy um pouco inferior, porém uma estimativa da orientação bem semelhante, portanto podemos concluir que a estratégia desenvolvida é válida para o campo completo porém ela apresenta uma sensibilidade muito maior em relação ao posicionamento a partir de longas observações mais pobres, onde se considerando apenas metade do campo como a área de trabalho, ao avistar as traves do gol poderíamos voltar a ter uma estimativa mais confiável. No campo completo não há como garantir que isso irá funcionar justamente por estarmos em uma área

de trabalho espelhada, podendo estar assim no mínimo em duas posições diferentes no campo para a melhor observação.

6.1 PERPESCTIVAS FUTURAS

O próximo passos para o desenvolvimento deste trabalho é a implementação deste algoritmo no NAO, onde para isso, é necessário avançar nos processos de detecção das features e ainda fazer uma adaptação do algoritmo proposto para que funcione baseado no processamento limitado da plataforma, onde alguns ajustes no tempo em que o filtro irá fazer a estimação também deverá ser revisto. Em relação ao processo de estimação o próximo objetivo é estudar novas maneiras de realizarmos o casamento das observações, como por exemplo, usar a distância de Mahalanobis, metodo esse que não foi usado neste trabalho por ainda não trabalhar-se com as incertezas associadas as observações. Também é possível trabalhar em novas estratégias para contornar a dificuldade de estimação após longos períodos de observação das features mais pobres, tentando caracterizar esses pontos e lhes atribuir valores.

O simulador é uma ferramenta poderosa, que permite que testemos algumas situações com uma facilidade muito maior. Porém ele apresenta certas limitações quanto as suas funções de movimentação, portanto, no futuro, pode-se trabalhar a implementação de um modelo de marcha mais próximo do real, levando em consideração o escorregamento por exemplo. Também seria interessante criar mais funções de movimentação, onde até mesmo controlá-lo pelo teclado ou joystick seria algo válido, podendo assim testar rapidamente o caminho desejado. Outro trabalho de grande contribuição seria a implementação das análises em tempo real, e não offline como feito neste trabalho, conferindo assim um maior dinamismo as análises.

A visão computacional é um processo importante, onde muitos elementos envolvidos foram encapsulados no simulador por se tratar de uma informação advinda de outro módulo do jogador. Porém, pensando na expansão do projeto, seria de grande ajuda a implementação de um sistema que simulasse as informações das câmeras do robô em termos dos pixels u e v, podendo assim auxiliar o grupo envolvido nos processos de visão computacional nos processos de detecção e estimação de distância até as features, dispensando em um primeiro momento o uso do robô e todo o aparato necessário para montar o campo.

[1] RoboCup Federation Official Website, disponível em: http://www.robocup.org/, último acesso em 22/11/2016.

[2] Cox, I., J. “Blanche - An Experiment in Guidance and Navigation of an Autonomous Robot Vehicle”, Em IEEE Transactions on Robotics and Automation, 1991;

[3] Leonard, J. J., Durrant-Whyte., H. F. “Mobile Robot Localization by Tracking Geometric Beacons”, Em IEEE Transactions on Robotics and Automation, 1991;

[4] Whelan, T., Studli, S., McDonald, J., Middleton, R. H., “Efficient Localization For Robot Soccer Using Pattern Matching”, Department of Computer Science, NUI Maynooth - Maynooth, Irlanda e Hamilton Institute, Nui Maynooth, Irlanda, 2011;

[5] Whelan, T., Studli., S., McDonald, J., and Middleton, R. H., “Line Point Registration: A Technique For Enhancing Robot Localization in a Soccer Environment” Em Proc. RoboCup Symposium, 2011.

[6] Sánchez, E. M., Alcobendas, M. M., Noguera, J. F. B., Bilabert, G. B., Ten, J. E. S. “A Reliability-Based Particle Filter For Humanoid Robot Self-Localization in RoboCup Standard Platform League”, Universidade Politécnica de Valencia, Valencia - Espanha, 2013;

[7] Anderson, P., Hunter, Y., Hengst, B., “An ICP Inspired Inverse Sensor Model with Unknown Data Association”, Em IEEE Transactions on Robotics and Automation, 2013;

[8] Carvalho, M. P. “Controle de Movimentação de Humanóide em Tempo Real por Teleoperação”, Trabalho de Graduação em Engenharia Mecatronica, Universidade de Brasília - Brasília, Brasil, 2014,;

[9] Documentação Oficial da Aldebaran, fabricante do robo humanoide NAO, disponível em: http://doc.aldebaran.com/, acessado em 10/08/2016;

[10] Rath, C., “Self-localization of a biped robot in the RoboCup Standard Platform League Domain”, Dissertação de Mestrado, Graz University of Technology - Graz, Austria, 2010;

[11] Bernardes, M. C., “Controle Servo-Visual para Aproximação de Portas por Robos Móveis Equipados com Duas Cameras”, Dissertação de Mestrado, Universidade de Brasília - Brasília, Brasil, 2009;

[12] Fraqueira, T. C., “Um estudo sobre Quatérnios e sua Aplicação em Robótica”, I-SBAI, Pontífice Universidade Católica de Minas Gerais - Belo Horizonte - Brasil, 1993;

[13] Thrun, S., Burgard. W., Fox, D., em Probabilistic Robotics, 2005;

[14] Biblioteca de Robótica para operação com Quatérnios DQRobotics, disponível em: https://sourceforge.net/p
, último acesso em: 17/08/2016;

[15] Camera Calibrate with OpenCV, disponível em: http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_ca
último acesso em: 23/11/2016;

[16] Find distance from camera to object/marker using Python and OpenCV, disponível em: http://www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/, último acesso em: 17/08/2016;